2005

High performance constraint satisfaction problem solving: State-recomputation versus state-copying.

Jigang Yang
University of Windsor

High Performance Constraint Satisfaction Problem Solving: State-recomputation versus State-copying

by

Jigang Yang

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2004

Your file Votre référence
ISBN: 0-494-04951-0
Our file Notre référence
ISBN: 0-494-04951-0

Abstract

Constraint Satisfaction Problems (CSPs) in Artificial Intelligence have been an important focus of research and have been a useful model for various applications such as scheduling, image processing and machine vision. CSPs are mathematical problems that try to search values for variables according to constraints. There are many approaches for searching solutions of non-binary CSPs. Traditionally, most CSP methods rely on a single processor. With the increasing popularization of multiple processors, parallel search methods are becoming alternatives to speed up the search process. Parallel search is a subfield of artificial intelligence in which the constraint satisfaction problem is centralized whereas the search processes are distributed among the different processors.

In this thesis we present a forward checking algorithm solving non-binary CSPs by distributing different branches to different processors via message passing interface and execute it on a high performance distributed system called SHARCNET. However, the problem is how to efficiently communicate the state of the search among processors. Two communication models, namely, state-recomputation and state-copying via message passing, are implemented and evaluated. This thesis investigates the behaviour of communication from one process to another. The experimental results demonstrate that the state-recomputation model with tighter constraints obtains a better performance than the state-copying model, but when constraints become looser, the state-copying model is a better choice.

Acknowledgements

First of all, I am most grateful to Dr. Scott Goodwin for introducing me to the topic of constraint satisfaction problems. Dr. Scott Goodwin has not only given me his valuable guidance on this thesis, but has provided his support through research grants as well.

I also appreciate the members of my committee, Dr. M. Hlynka and Dr. X. J. Chen, not only for spending precious time to read through my drafts, but also for their detailed comments and advice in completing this thesis.

The School of Computer Science, University of Windsor, has provided me with a great deal of support. I would like to thank my fellow students, Mingyan Huang, Bob, Eric and Lucy Lu for their help and useful feedback during my research. I would like to thank Bob in particular for his precious time in reading an earlier draft of this thesis.

I am grateful for the support from my family during my graduate study in University of Windsor. My special thanks go to my uncle Howard. Without his support this thesis could never have been completed.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Many problems in the area of artificial intelligence can be treated as constraint satisfaction problems (CSPs). The constraint satisfaction problem is a well-known terminology for various techniques used in artificial intelligence and related disciplines. The study of constraint satisfaction problems was initiated by Montanari in 1974 [Mont74]. It was realized that the same general framework was applicable to a much wider class of problems. CSPs provide a convenient way to represent and solve certain kinds of problems. Over the past years CSPs have become one of the most versatile mechanisms for representation and modeling as well as solving complex relationships in real life applications such as design and configuration, diagnosis, debugging, graph coloring, decision support, scheduling, planning, resource allocation, and supply chain management. Therefore, CSPs warrant further study. In short, they form a powerful framework for solving general problems.

1.1 CSP Definition

A Constraint Satisfaction Problem is a problem which consists of:

- A set of variables,
- Each variable is associated with a finite set of possible values,
- A set of constraints restricting the values that the variables can simultaneously take.

Three important factors in a constraint satisfaction problem are variables, domains with each variable, and constraints among these variables. With the CSP representation,

different approaches can be used to solve it within the context of the application. The constraint satisfaction problems belong to the class of NP complete problems, which in general, require time that grows exponentially with the problem size. It is unknown whether there are any faster algorithms or not. Normally there are three main solving techniques: problem reduction, search, and solution synthesis. The search method is intended to enumerate the search space in order to find solutions. Many algorithms have been proposed, most of which are variations of the basic backtracking algorithm. The problem reduction uses constraint propagation to eliminate redundant values from the original problem definition so that the size of the search space decreases. Although problem reduction alone does not normally produce solutions, it can be efficient when used together with search or problem synthesis methods. The solution synthesis was initially proposed by Freuder [Freu78]. It constructs and extends partial solutions in order to generate the set of all solution tuples. These approaches are the most studied in CSP research. This thesis mainly concerns the forward checking algorithm of the search method which is a variation of the basic backtracking algorithm.

In the early research of CSPs, many approaches were developed which focused on binary CSPs, where each constraint is of arity at most two. The arity is the number of variables in a constraint. Unfortunately, in real life problems, non-binary CSPs, known as general CSPs, exist frequently in applications such as timetabling, routing or scheduling. A non-binary constraint satisfaction problem consists of at least three variables. It is well known that any non-binary constraint satisfaction problem can be translated into an equivalent binary CSP. Two translations are known: the dual graph translation and the hidden variable translation [RDP90]. In this thesis, we intend to solve non-binary CSPs using the dual graph translation technique.

1.2 Motivation

There are a large number of approaches to solving CSPs and many extensions and improvements of these approaches have been studied and developed. However, most non-binary CSP methods rely on a single processor. Over the past years, due to advances in parallel computing, parallel processing has become more widely available. Parallelization is a good candidate to obtain solutions faster on CSPs solution methods.

A parallel search method is a subfield of artificial intelligence in which the constraint satisfaction problem is centralized while the searches are distributed among different processors. In order to speed up the traditional search approaches, search methods can be performed in parallel by partitioning the search tree into many disjunct parts that can be explored concurrently or by splitting the initial CSP into a collection of many easier subproblems.

Furthermore, in order to get an efficient parallel tree search, a load balancing strategy is taken into account. The objective of load balancing is to make all the processors busy without an improper overhead. Load balancing distributes jobs over a network of computer systems or clusters, thus increasing throughput without having to require additional or faster computer hardware.

1.3 Central Theme

In this thesis we design and implement the forward checking algorithm for solving non-binary CSPs based on parallelism via message passing interface (MPI) and load balancing strategy. We execute the procedure on a high performance distributed system called Shared Hierarchical Academic Research Computing Network (SHARCNET). The main theme of this thesis is the comparison of the behaviour of communication among processors. One communication model is state-recomputation where a message

containing an oracle is sent to determine the search path. Another model is state-copying where a message is sent which copies the current search state. From the experimental results we evaluate and compare the performance of these options on multiple processors.

1.4 Outline

The outline of this thesis is structured as follows: Chapter 2 provides some background relating to CSPs and techniques for solving CSPs, the basis of parallel computing including the message passing interface (MPI) and knowledge of SHARCNET. Chapter 3 discusses the details of the forward checking algorithm solving a non-binary CSP by state-copying communication and state-recomputation communication in the distributed environment. Chapter 4 contains some empirical studies and experimental results. Chapter 5 contains the conclusion and future work. At the end, the bibliography is given.

Chapter 2 Background

Constraint satisfaction problems (CSPs) can be traced to the 1970s but their widespread study, research and application started since the end of 1980s. Over the past years a number of approaches to constraint satisfaction problems have been developed. The techniques in CSP solving can basically be classified into three categories: (1) problem reduction, which is to filter or pre-process; (2) search, which is to find solutions; (3) solution synthesis, which can be seen as search algorithms which explore multiple branches simultaneously and in which the solutions of the branches are integrated into one solution. These techniques will interact with each other. In order to handle large-scale real life problems, many advanced techniques of CSP should be studied to meet these problem requirements. With the development of parallel computing, the adaptation of CSPs to distributed environments will warrant further study. In this chapter we will introduce related CSP background and basic knowledge of parallel computing.

2.1 Preliminaries

Throughout this thesis we follow the notations given below. Uppercase X_i represents a variable. Uppercase C_i is an individual constraint which has a set of satisfying tuples S_i covering the set of variables X_i. For example, the notation $C_{1,2,3}$ stands for a constraint involving $\{X_1, X_2, X_3\}$.

Definition 2.1 A **constraint satisfaction problem** (CSP) is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take [Tsan93]. A CSP is a tuple $<X, D, C>$ where $X = \{X_1,\ldots,X_n\}$ is a set of variables, $D = \{D(X_1),\ldots,D(X_n)\}$ is a set of finite domains, and C is a set of constraints that restrict certain simultaneous object assignments. So each $X_i \in X$ has a corresponding discrete domain D_i which can be instantiated. Every element $C_i \in C$ is a constraint over a subset of variables of X. It contains tuples of objects that are not allowed to be assigned simultaneously. Therefore, the solution of a CSP is to assign to each $X_i \in X$ an object from $D(X_i)$, such that no $C_i \in C$ is violated.

Consider the map-coloring CSP example in Figure 2.1 [Kuma92]. Here the map has four regions which are to be coloured by red, blue or green. Figure2.2 is the equivalent constraint graph for this problem. Each area is a variable and the domain of each variable is the given set of colors such as red, blue and green. According to the rule, there is a constraint between the corresponding variables which does not allow the same value to be assigned to these two variables.

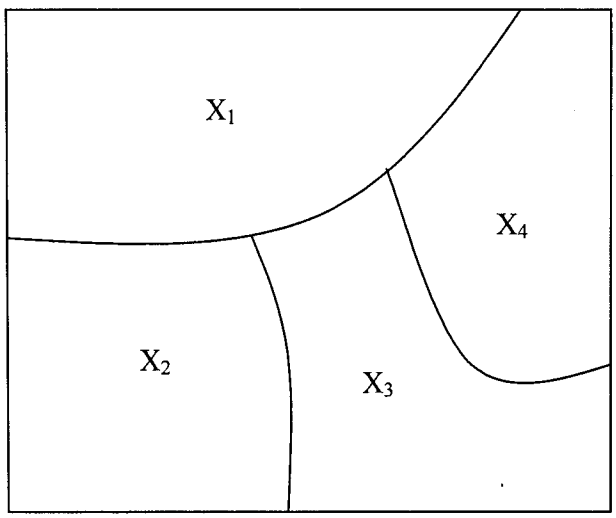


Figure 2.1 A map-coloring problem

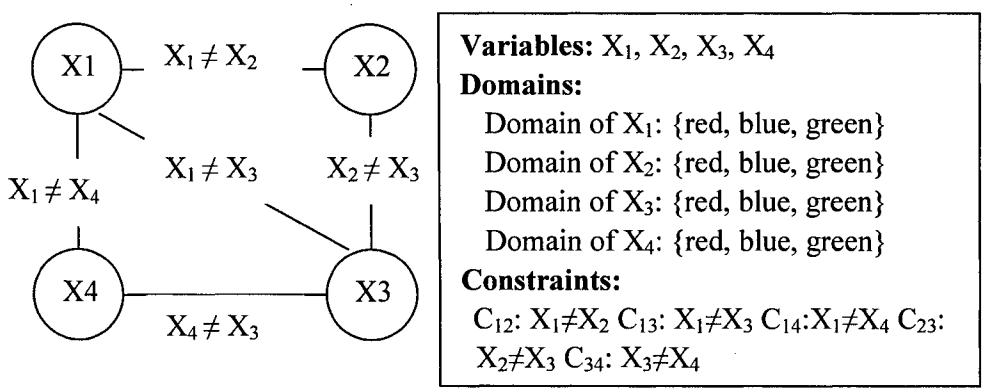


Figure 2.2 An equivalent constraint graph for Figure 2.1

Definition 2.2 Given a constraint satisfaction problem as above,

1) A **solution** is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied simultaneously.

2) The number of variables in a constraint is called the **arity** of the constraint.

3) A CSP in which each constraint is of arity at most two is called a **binary CSP**.

4) A CSP where the arity is greater than two is called a **non-binary CSP**.

5) The constraints of a non-binary CSP are usually represented in one of two ways: the specification of a constraint by explicitly listing all the valid combinations of values for the tuples of variables is referred to as **extensional representation**; For example, if there are three variables X_1, X_2 and X_3, domain size is {1, 2, 3} and the constraints are C_{123} relating to variables $\{X_1, X_2, X_3\}$ and C_{23} relating to variables $\{X_2, X_3\}$, then the tuples may be {1, 1, 1}, {1, 1, 3}, {1, 2, 2} and {2, 3, 3} for C_{123}; {2, 2}, {2, 3} and {3, 1} for C_{23}. On the other hand, the specification of a constraint which implicitly defines the satisfying tuples by a mathematical expression such as equal, less-than, greater-than etc. is referred to as **intensional representation**. For example, we could have the

variables, X_1, X_2, and X_3 with domain {1, 2, 3}, and a constraint $C_{1,3}$ given in intensional form as $X_1 > X_3$. Although the constraints of real problems are not often represented in an extensional way, this way emphasizes that constraints do not need to correspond to the mathematical expressions.

6) The **tightness** of a CSP is measured by the number of solution tuples over the number of all distinct compound labels for all variables. In the map coloring example above, there are five constraints $\{C_{12}, C_{13}, C_{14}, C_{23}, C_{34}\}$ and the number of satisfied tuples of each constraint is 6, {(blue, red), (blue, green), (red, blue), (red, green), (green, red), (green, blue)}. The number of all distinct compound labels for all variables for each constraint is 9, {(blue, red), (blue, green), (blue, blue), (red, blue), (red, green), (red, red), (green, red), (green, blue), (green, green)} Thus, the tightness of each constraint is 6/9 = 66.667%.

2.2 Example

Consider the following constraint problem as a non-binary CSP with 4 variables, X_1, X_2, X_3, and X_4. The domain sizes of the variable are $D\{X_1\} = \{1,2,3,4,5\}$, $D\{X_2\} = \{1,2,3,4,5\}$, $D\{X_3\} = \{1,2,3,4,5\}$ and $D\{X_4\} = \{1,2,3,4,5\}$. The constraints are: $C_{1,2,3}$: $X_1 - X_2 = X_3$, $C_{1,4}$: $X_1 = X_4$, and $C_{2,3}$: $X_2 = X_3$; in this case, we use the extensional form and $S_{1,2,3} = \{(2,1,1),(3,1,2),(3,2,1),(4,1,3),(4,2,2),(4,3,1),(5,1,4),(5,2,3),(5,3,2),(5,4,1)\}$, $S_{1,4} = \{(1,1),(2,2),(3,3),(4,4),(5,5)\}$, and $S_{2,3} = \{(1,1),(2,2),(3,3),(4,4),(5,5)\}$.

Any non-binary constraint satisfaction problem can be translated into an equivalent binary CSP. Here are two methods: the dual graph translation and the hidden variable translation. Both methods change the set of variables of the original problem to a new one. The dual graph was derived from the relational database community and was presented by Dechter and Pearl [DP88].

Definition 2.3 ([DP89]) A **dual constraint graph** is an undirected graph where each node represents a constraint. There is an arc between any two nodes sharing a common variable. The arcs are labeled by the shared variables. The dual constraint graph can be used to transform any non-binary CSP into a special type of binary CSP.

Here we explain how the dual graph method transforms a non-binary CSP into a binary CSP. According to the non-binary CSP example above, we construct a new binary CSP. First, we construct the new corresponding variables. The constraint $C_{1,2,3}$ will be changed to the new dual node Y_1. The constraint $C_{1,4}$ will be changed to the new dual node Y_2. The constraint $C_{2,3}$ will be changed to the new dual node Y_3. We next construct the constraints. We check any sharing variables between variables in the new dual graph. There is one variable, X_1, between Y_1 and Y_2. There are two variables, X_2, X_3, between Y_1 and Y_3. And there is no sharing variable between Y_2 and Y_3. The domain of each dual variable is the set of tuples that satisfy the constraint. Figure 2.3 shows a binary CSP resulting from the dual graph method.

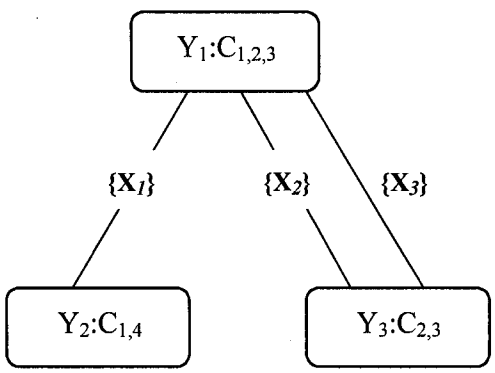


Figure 2.3 Binary CSP resulting from the dual graph method

In the figure 2.3, the dual variables are $\{C_{1,2,3}; C_{1,4}; C_{2,3}\}$, the dual constraints are $\{X_1, X_2, X_3\}$ and the domain size of the dual variables are

$D\{C_{1,2,3}\}=\{(2,1,1),(3,1,2),(3,2,1),(4,1,3),(4,2,2),(4,3,1),(5,1,4),(5,2,3),(5,3,2),(5,4,1)\}$

$D\{C_{1,4}\}=\{(1,1),(2,2),(3,3),(4,4),(5,5)\}$ and $D\{C_{2,3}\}=\{(1,1),(2,2),(3,3),(4,4),(5,5)\}$

Definition 2.4 The **hidden variable method** retains all the variables of the original CSP and adds new nodes which represent the constraints in the hidden representation. The hidden variable method is also referred to as **hidden encoding**.

2.3 Solution Techniques

It is well known that solution techniques for CSPs can be divided into three categories: problem reduction, search and synthesis. In this section we focus on studying of these techniques.

2.3.1 Problem Reduction

Problem reduction is a class of techniques for transforming a CSP into problems which are easier to solve by reducing the size of the domains and constraints. It is the process of removing values from the domain and tightening constraints in the CSP without ruling out solution tuples from a CSP [Tsan93]. Its definition is as follows: A problem P = (X, D, C) is reduced to P' = (X', D', C') if P and P' are equivalent, where X is a finite set of variables $\{X_1,X_2,\ldots,X_n\}$; D is a set of finite domains and C is a set of constraints that restrict certain simultaneous object assignments. Every variable domain in D' is a subset of its domain in D, and C' is more restrictive than or as restrictive as C.

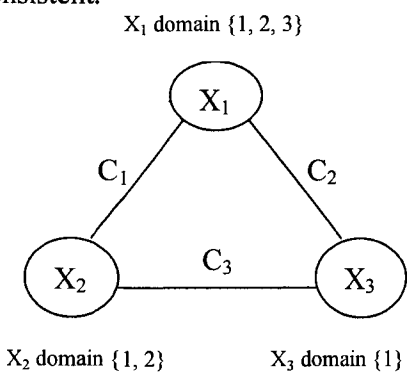
Problem reduction is often referred to as consistency maintenance or problem relaxation. Problem reduction can help to solve the CSP but it will not produce the solutions. However, it is useful when it is used together with search or problem synthesis methods. Using the problem reduction technique will obtain many benefits such as reducing the search space by reducing the domain size, detecting an insoluble problem and avoiding repeatedly searching futile subtrees. A. K. Mackworth [Mack77] defines three consistency techniques borrowing terminology from graph theory: Node Consistency (NC), Arc Consistency (AC), and Path Consistency (PC).

2.3.1.1 Node Consistency

Node Consistency (NC) is the simplest consistency technique. A CSP is node-consistent if and only if all values in its domain satisfy the constraints on that variable. The algorithm which conducts Node Consistency over a CSP is called the NC algorithm. The NC algorithm checks each element in each domain and checks if that value satisfies the unary constraint of the variable or not. All values which fail to satisfy the unary constraints will be removed from the domains.

2.3.1.2 Arc Consistency

Arc(X_i, X_j) is arc consistent if for every value x of the current domain of X_i there is some value y in the domain of X_j such that $X_i = x$ and $X_j = y$ is permitted by the binary constraint between X_i and X_j. The concept of arc consistency is directional, i.e., if an arc(X_i,X_j) is consistent, then it does not automatically mean that arc(X_j,X_i) is also consistent.



Variables: X_1, X_2, X_3
Domains:
Domain of X_1: {1, 2, 3}
Domain of X_2: {1, 2}
Domain of X_3: {1}
Constraints:
C_{12}: $X_1 \neq X_2$;
C_{13}: $X_1 \neq X_3$;
C_{23}: $X_2 \neq X_3$:

Figure 2.4 A constraint graph with arc consistency

In Figure 2.4 we can see that arc(X_3, X_1) is consistent because 1 is the only value in the domain of X_3 and for $X_3 = 1$ there is at least one value for X_1 that satisfies the constraint between X_3 and X_1, i.e., $X_1 = 2$. However, the arc consistency is directional. For example, the arc(X_3, X_1) is consistent but the arc(X_1, X_3) is not consistent because for

$X_1 = 1$, there is no value in the domain of X_3 which is permitted by the constraint between X_3 and X_1.

Arc(X_i, X_j) can be made consistent by simply eliminating those redundant values from the domain of X_i. The algorithm, REVISE, makes an arc consistent [Bart98]. For arc(X_i, X_j), REVISE will check each value in the domain of X_i. If for a value x in the domain of X_i, there is no value in the domain of X_j to satisfy the constraint between X_i and X_j, then value x will be removed from the domain of X_i. If any value is removed from the domain of X_i, REVISE will return the value of true, which means this arc is originally not consistent and this arc is changed to be consistent after REVISE.

The arc consistency can potentially eliminate more redundant values from the domains than NC. But if we use REVISE on every arc, this is not enough. Pruning the domain may make some already revised arcs inconsistent again. Thus, the arc revisions will be repeated to all constraints in both directions until no changes are made. Mackworth proposed algorithm AC-1 in 1977 [Mack77]. The algorithm is as follows: The algorithm AC-1 repeatedly runs on each arc in the constraint graph. In every cycle, the algorithm will check if any arc is made during the iteration. If return value is still true, then AC-1 will launch other iterations in the constraint graph. AC-1 is simple but inefficient, because one revision causes all arcs to be revised on next iteration, even though only a small number of them are really affected by this revision. Mackworth presented an improvement on AC-1 which is called AC-3 that eliminates this drawback. Compared with algorithm AC-1, AC-3 performs a revision on one cycle leading to a revision on the next cycle of only those arcs that might be affected. In practice, it is probably the most widely used consistency algorithm. The AC-3 algorithm is as follows: it sets up a queue containing all the arcs of a constraint problem. When the queue is not empty, AC-3 selects and removes an arc(X_i, X_j) from the beginning of the queue. If arc(X_i, X_j) is not consistent, and if (X_k, X_i) is not in the queue, add (X_k, X_i) to the end of the queue with k different from i and j in the constraint graph. This procedure continues until the queue is empty. Many variations and improvements of AC-3 have

been developed. From Mohr and Henderson, AC-4 was developed in 1986 [MH86]. The elements in the queue are variable-value pairs. For each arc from x to y, and for each value V_x in the domain of x, keep a counter of the number of values in the domain of y that it is consistent with. If the counter goes to zero, remove V_x from the domain of x and update the counter for y [HL88]. AC-4 needs a special data structure to remember pairs of consistent values of incidental variables, thus, it is less memory efficient than AC-3. The AC-6 algorithm [Bess94] uses the same principles as AC-4. It avoids a lot of expensive checks and storage requirements of AC-4 by the principle of minimal support. The AC-7 algorithm [BFR95] can exhibit significant savings over previous arc-consistency algorithms and it can be applicable to any binary CSP [Naga01]. There exist several arc consistency algorithms beginning from AC-1 and ending at AC-7. These algorithms are based on the repeated revisions of arcs until a consistent state is reached or some domain becomes empty. The most popular among the algorithms are AC-3 and AC-4. Other algorithms, AC-5, AC-6, and AC-7, are not used as frequently as AC-3 or AC-4.

2.3.1.3 Path Consistency

In addition to arc consistency, other types of consistency have been defined for binary CSPs. Path consistency is stronger than arc consistency. Even more inconsistent values can be deleted by path consistency (PC) techniques. It is useful in a backtracking search where it can be used as a pre-processing algorithm and as an algorithm that can be interleaved with the backtracking search. Path consistency requires that for every pair of values of variables X, Y satisfying a binary constraint, there exists a value for each variable along some path between X and Y such that all binary constraints in the path are satisfied. Of course, this increase in the pruning power of path consistency does not come for free. The computational complexity of a path consistency algorithm is greater than that of arc consistency algorithms.

2.3.2 Search

Search is the most studied approach in CSP research. Some approaches use constraint propagation to reduce the original problems. Others use backtracking to directly search for possible solutions. CSP can always be solved by the standard backtracking algorithm, although at substantial cost. There are a lot of search techniques and algorithms for solving CSPs but most of them are variations of the backtracking algorithm.

2.3.2.1 Chronological Backtracking

Chronological backtracking, often just called backtracking (BT), is a common search technique. It is probably the most widely used systematic search algorithm and is basically a depth-first search. The algorithm was first presented by Bitner and Reingold [BR75]. Figure 2.5 shows the control flow of backtracking. Within the CSP context, the basic operation is to select one variable at a time, and consider one value for it at a time, and make sure that the new one is compatible with all the constraints. If the current variable with the new value violates a certain constraint, then an alternative value, when available, is selected. If all the variables are labeled, then the problem is solved. The algorithm continues until either a solution is found or all the combinations of values have been tried and have failed.

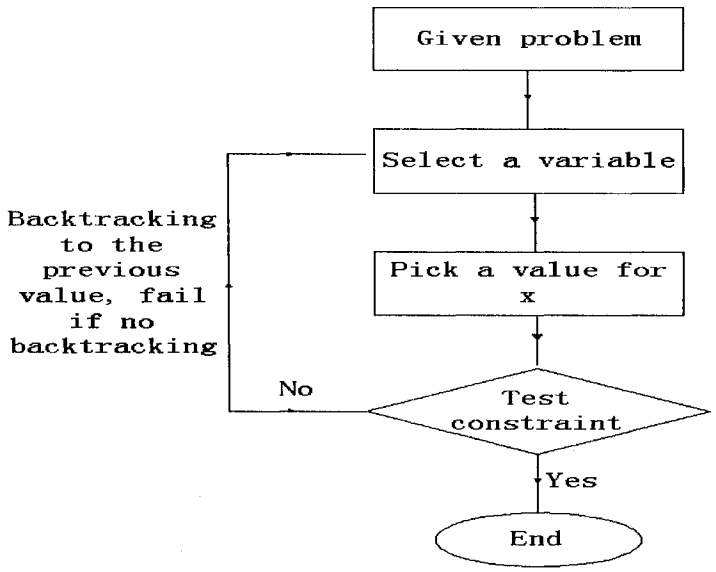


Figure 2.5 The control flow of the chronological backtracking algorithm

Chronological backtracking has some drawbacks such as redundant work and late detection of the conflict. Its efficiency depends on the order of exploring alternatives-not on their logical relationships. Backtracking is one algorithm for solving CSPs, but it is not the best.

2.3.2.2 Other Search Algorithms

The forward checking (FC) algorithm for solving constraint satisfaction problems is a popular and successful alternative to backtracking. Forward checking combines a backtracking search with a very limited form of arc consistency maintenance [BG95]. The forward checking algorithm makes the consistency checks between the current variable and the remaining variables not yet instantiated. The domains of the future variables are filtered in such a way that all values inconsistent with the current instantiation are removed. Forward checking is very efficient because of its ability to discover inconsistencies early and reduce the search space. The main difference between forward checking and backtracking is that forward checking checks the domain each time after one value is selected. If any domain is reduced to empty, then

forward checking will reject the current assigned value immediately. Figure 2.6 shows the control flow of the forward checking algorithm [Tsan93].

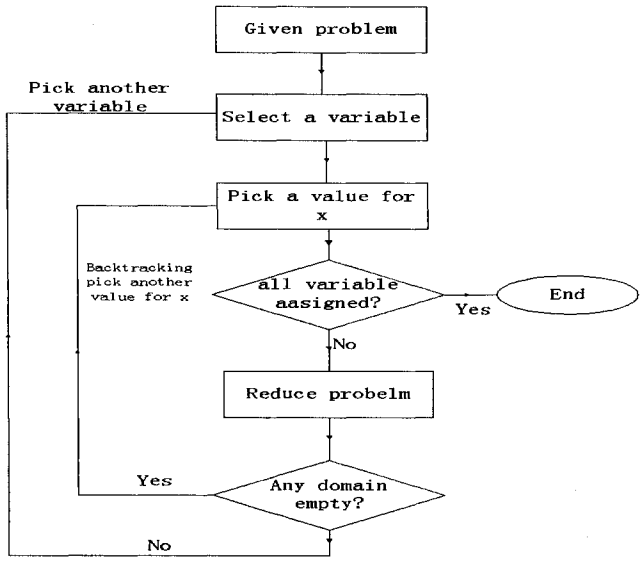


Figure 2.6 The control flow of forward checking algorithm

The following example in Figure 2.6 is traced by the forward checking (FC) algorithm in solving the 4-Queens problem.

1. First of all, FC algorithm selects a variable X_1 and picks the value 1 for X_1;
2. FC algorithm does the function called reduce problem; It removes $X_2 = 1$ and $X_2 = 2$. Because $X_1 = 1$ and $X_2 = 1$ violate the constraint; $X_1 = 1$ and $X_2 = 2$ also violate the constraint; And it removes $X_3 = 1$, $X_3 = 3$, $X_4 = 1$, and $X_4 = 4$;
3. FC algorithm selects a variable X_2 and picks the value 3 for X_2;
4. FC calls the function of reduce problem; It removes $X_3 = 2$, $X_3 = 4$ and $X_4 = 3$. It does backtracking;
5. FC picks the value 4 for X_2;
6. FC calls the function of reduce problem; it removes $X_3 = 4$ and $X_4 = 2$. Because these violate the constraint;
7. FC selects a variable X_3 and picks the value 2 for X_3;
8. FC calls the function of reduce problem; it removes $X_4 = 3$. It does backtracking;
9. FC algorithm picks the value 2 for X_1;
10. FC does the function of reduce problem; it removes $X_2 = 1$, $X_2 = 2$, $X_2 = 3$. All of

these violate the constraint; And it removes $X_3 = 2$, $X_3 = 4$, and $X_4 = 2$;

11. FC selects X_2 and gives the value 4 to X_2;
12. FC calls the function of reduce problem and removes $X_3 = 3$ and $X_4 = 4$. All of these violate the constraint;
13. FC selects X_3 and gives the value 1 to X_3;
14. FC calls the function of reduce problem and removes $X_4 = 2$. It violates the constraint;
15. FC selects X_4 and gives the value 3 to X_4;
16. All of the variables have been assigned, so the FC algorithm is terminated. Finally we get the result as shown in Figure 2.7.

Therefore, the solution from the FC algorithm is $\{X_1 = 2, X_2 = 4, X_3 = 1, X_4 = 3\}$.

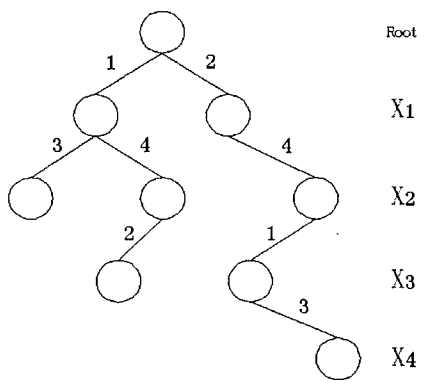


Figure 2.7 4-Queens problem solved by forward checking algorithm

In addition to forward checking, there are many lookahead algorithms such as Partial lookahead and Full lookahead [HE80]. With these approaches, more constraint propagation will result in less searching, but the overall cost may be higher, as the processing at each step will be more expensive. Actually, in some cases the full look ahead may be more expensive than simple backtracking. That is the reason why forward checking and simple backtracking are still used in applications. Furthermore, during the search, CSPs allow learning from experience and analyze the reason for failure in order to avoid making the same errors repeatedly in the future. There are also other algorithms such as Backjumping, Backchecking and Backmarking [Gasc77,

Gasc79, Nide83a, Nide83b]. In summary, these algorithms are divided into general search methods, lookahead methods and gather information while searching methods. In addition, stochastic search methods such as hill-climbing are becoming increasingly popular and crucial for solving the combinatorial explosion problem in CSP. In many situations, a delay in decision could not be allowed and a timely response is needed. The stochastic search method sacrifices completeness for speed. It is a class of search approaches guided by heuristics. There is no universally best algorithm for all problems and different problems can be solved most efficiently by different algorithms.

2.3.3 Solution Synthesis

As mentioned at the beginning of this chapter, there is another approach, called solution synthesis, applied to CSP besides search and problem reduction. Solution synthesis is an approach used to find all solutions to a CSP. That means all assignments of values to variables that satisfy the problem's constraints are produced by a solution synthesis algorithm. The solution synthesis algorithm was first presented by Freuder [Freu78]. It is applicable to CSPs with general constraints. The basic idea is to create a lattice, called an MP-graph, in which every node contains the set of all legal tuples for a unique subset of variables [Tsan93]. Besides this, there are two other solution synthesis algorithms. One is the Invasion Algorithm, proposed by Seidel. Its basic idea is to search the partial graphs of the invasion. Although it can be extended to deal with general constraints, it is especially suitable to binary constraint satisfaction problems [Seid81]. Another synthesis algorithm is the Essex Algorithm. It is also applicable to binary constraint problems [TF90]. All three solution synthesis algorithms are limited to CSPs in which all the solutions are required. These in general are more useful for tightly constrained problems.

There are many approaches to solving CSPs approaches. However, most non-binary CSP methods run on a single processor. In order to speed up the traditional search

approaches, search methods can be performed in parallel. In the next section we introduce some basic ideas about parallel computing.

2.4 Introduction to Parallel Computing

Traditionally, parallel computing is the simultaneous use of processing units that cooperate to solve computational problems quickly. It can provide higher computational capability than the fastest serial computing. It not only solves problems that do not fit on a single CPU's memory space, but solves problems that cannot be solved in a reasonable time as well [CSG98, HX98]. Until recently, Flynn's taxonomy was commonly used to classify parallel computers into one of the four basic types as follows [Flyn72]: 1. Single instruction, single data (*SISD*); it is a serial (non-parallel) computer. 2. Single instruction, multiple data (*SIMD*); all processing units execute the same instruction at any given clock cycle, each processing unit can operate on a different data element. 3. Multiple instruction, single data (*MISD*); few actual examples of this type of parallel computer exist. 4. Multiple instruction, multiple data (*MIMD*); this is the most common type of parallel computer. Another way to classify modern parallel computers is by their memory model: Shared memory, distributed memory and hybrid of the above two. The shared memory system means that the different processors share a common memory. It does not require the processors to communicate data and control information by explicitly passing messages between the memory modules. It can be divided into two main classes based upon memory access time: Uniform memory access (*UMA*) and Non-uniform memory access (*NUMA*). The main advantage is that global address space provides user-friendly programming to memory and the disadvantage is the lack of scalability between memory and CPU. Unlike a shared memory system, the distributed memory system means that each processor has its own dedicated memory. Its architecture is scalable and has the highest potential performance, but requires that the users incorporate explicit passing of messages between processor nodes for both data and control information. The hybrid

distributed-shared memory system is used in the largest and faster computers in the world today and employs both shared and distributed memory architectures [Nilm91].

In parallel programming models, there are several models in common use: Shared memory, Threads, Message passing [Mess94, Mess97], Data parallel, and Hybrid. The advantage of the shared memory model from the programmer's point of view is that program development can often be simplified and there is no need to specify explicitly the communication of data between tasks. The threads model allows a single process to have multiple, concurrent execution paths. Threads are commonly associated with shared memory architectures and operating systems. There are two different implementations of threads: POSIX Threads and OpenMP. Apart from the above, designing parallel programs is the main topic in parallel computing. In practice, considering a parallel program may include some or all of the following [Buyy99]:

1. Partitioning or decomposition: this is the first procedure in designing a parallel program. There are two ways to partition the work: domain decomposition and functional decomposition.
2. Communications: there are several factors to be considered such as cost of communications, latency, bandwidth and synchronization vs. asynchronous.
3. Data dependencies: this is important to parallel programming. Communicating required data at synchronization points and synchronizing read/write operations between tasks are two ways to handle data dependencies.
4. Load balancing and Granularity: this refers to partitioning the work equally and using dynamic work assignment to achieve load balance. Granularity is the ratio of computation to communication. There is fine-grain parallelism and coarse-grain parallelism. For the latter, it is harder to load balance efficiently.
5. Input/Output: I/O operations are generally regarded as a bottleneck to parallelism. Fortunately, some parallel file systems and parallel I/O programming interfaces are available.
6. Limits and costs of parallel programming: it needs to consider speedup, complexity, portability, resource requirements, and scalability among other factors.

Parallel computing has made a tremendous impact on a variety of areas. It has been employed with great success everywhere. The enormous computing power and large memory space of parallel computers has opened a new horizon for researchers of this area.

2.5 Message Passing Interface (MPI)

As mentioned above, message passing is used to exchange data between parallel tasks, and to synchronize the tasks. The message passing model assumes a group of processes that have only local memory but are able to communicate with other processes by sending and receiving messages [GLS99, Pach98, GSN+98]. Message passing is one of the most powerful and widely used paradigms for parallel computing. The message passing standard is Message passing interface (MPI). MPI is a specification for programmers of message passing libraries. It is based on a message passing parallel programming model. It is a relatively new tool in parallel programming. In November 1993, the draft MPI standard was presented at the Supercomputer 93 conference. The final draft was released in May, 1994. It is widely used in parallel programming today. The second version 2.0, the most recent version, has been implemented by a majority of hardware vendors. It contains significant enhancements over version 1.x, such as one-sided communication, dynamic process creation, and extended collective operations. In addition to MPI implemented by hardware vendors, there are several publicly available MPI implementations developed by government research labs and universities. For example, the MPICH [GL96] is provided by Argonne National Lab and LAM-MPI is distributed by Indiana University. There are several reasons for using MPI.

1. Standardization: MPI is the only standard message passing library. It has replaced all previous message passing libraries.
2. Portability: There is no need to modify the source code when the applications are planted to a different platform.

3. Availability: it can be used in FORTRAN and C. A variety of implementations are available.

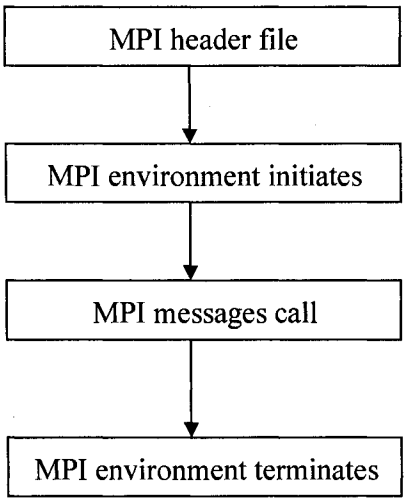


Figure 2.8 MPI program structure

Figure 2.8 points out the general MPI program structure. MPI has many routines, but most of the commonly used ones are described below:

1. MPI_Init: initializes the MPI execution environment. This function must be called in every MPI program before any other MPI functions and only once in an MPI program.
2. MPI_Comm_size: determines the number of processes within a communicator.
3. MPI_Comm_rank: initially each process will be assigned a unique integer rank beginning at zero and contiguous.
4. MPI_Send: it sends a message.
5. MPI_Recv: it receives a message.
6. MPI_Finalize: terminates the MPI execution environment. It should be the last MPI routine in every MPI program.

In MPI programs, we must call MPI_Init prior to any other MPI calls because this call sets up the MPI environment. At the end of the program, we should call MPI_Finalize to close the MPI environment, and after this call, no more MPI calls are allowed. Let us take a look at the example (Figure 2.9) below to explain the usage of MPI routines.

```
1.  #include <mpi.h>
2.  #include <math.h>
3.  #include <stdio.h>
4.  #include <stdlib.h>
5.  void main(int argc, char * argv[]) {
6.      int i, size, myid;
7.      int tag=0;
8.      double starttime, endtime, elapsed;
9.      int message;
10.     MPI_Status status;
11.     MPI_Init(&argc, &argv);
12.     MPI_Comm_rank(MPI_COMM_WORLD, &myid);
13.     MPI_Comm_size(MPI_COMM_WORLD, &size);
14.     Starttime = MPI_Wtime();
15.     for(i=0;i<100000;i++){
16.     if(myid == 0) {
17.     MPI_Send(&message,1,MPI_INT,1,tag,MPI_COMM_WORLD);
18.     MPI_Recv(&message,1,MPI_INT,1,tag,MPI_COMM_WORLD,&status);
19.     endtime = MPI_Wtime();
20.     elapsed = (endtime-starttime); }
21.     else {
22.       MPI_Recv(&message,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
23.       MPI_Send(&message,1,MPI_INT,0,tag,MPI_COMM_WORLD);
24.       }End if*/
25.     } /*End for*/
26.     if(myid==0) printf("The process %d elapsed time is%f\n",myid,elased);
27.     if(myid==1) printf("The process %d elapsed time is%f\n",myid,elased);
28.     MPI_Finalize();
29. }/*End main*/
```

Figure 2.9 The ping-pong program for data transfer rate

The program is well known as the ping-pong test. It tests the data transfer rate between two processors. Line 14 starts to count the beginning time. Line 16 is executed by one of the processors. Line 17 sends a message to the other processor and waits. Line 22 is executed by the other processor, receives the message, and then sends a message to the first processor. Line 18 receives the message and starts to count the ending time. The program gets the data transfer rate.

The program begins at MPI including the file "mpi.h" like other C library files. In the program, line 11, MPI_Init() initializes the MPI execution environment. Also, line 28,

MPI_Finalize() terminates the MPI execution environment. Message passing is done by lines 17, 18 and lines 22, 23, which are MPI_Send() and MPI_Recv(). These are basic functions in MPI. MPI_Send() function sends a message from one process to another process, and MPI_Recv() function receives a message from another process. The processes belong to the communicator. The default communicator is called MPI_COMM_WORLD. The syntax of MPI_Send() is: int MPI_Send(void *message, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm), where count is the number of messages, datatype is the type of message in MPI, dest is the rank of destination process, tag is used to distinguish among different message, and comm is the communicator which consists of all the running processes. The syntax of MPI_Recv() is: int MPI_Recv(void * message, int count, MPI_Datatype, int source, int tag, MPI_COMM, comm, MPI_Status * status). Line13, MPI_Comm_size(), is used to determine the number of processes within a communicator. Line 12, MPI_Comm_rank, is used to give the rank number for each process. If there are *n* processes executing the program, they will have ranks 0, 1, ..., *n*-1.

MPI is the specification of message passing, not a language or compiler. It is a reliable message exchange with a rather small overhead in a heterogeneous environment and portable to many high end platforms. However, sending and receiving data among distributed memory, may be expensive. If the amount of information that needs to be sent back and forth is large, it will slow down the performance.

Apart from MPI, there is the application program interface (API) in the shared memory architectures or platforms which is called OpenMP. It stands for *Open* specifications for *Multi Processing* via collaborative work with interested parties from the hardware and software industry, government and academia [OMP97]. It is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer. It is an API used with C/C++, FORTRAN for programming shared address space machines. The parallelization of OpenMP is designed for the SMP

programming paradigm, i.e. the machine should have a global address space. In some high-end platforms, OpenMP parallel programming can be mixed with the Message passing interface (MPI) library.

2.6 Load Balancing

In order to get an efficient parallel tree search, the load balancing strategy is the central issue. The objective of load balancing is to make all the processors busy without an improper overhead. Load balancing policy may be either static or dynamic. Static load balancing policy is generally based on information about the average behavior of system and decisions are independent of the actual current system state. A dynamic policy, on the other hand, reacts to the actual current system state. It is more flexible than a static one and can support irregular problems. Approaches to dynamic load balancing includes: load evaluation, profit determination, work transfer calculation, task selection, and migration [KR87].

In the static situation, for example, in the parallel search tree, the initial partitioning of the nodes in the search tree is performed before the search is initiated. The tree is divided into a number of subtrees: one subtree for each processor. Then each processor executes a sequential search on the subtree assigned to it. The communication is limited to the initial distribution of the subtrees. However, there is one main drawback: the times needed to process the subtree can totally differ. Therefore, some processors may be idle long before the program is finished. Figure 2.10 presents the drawback of static load balancing. In this graph, assume that there are four processors. The root node creates four nodes (A, B, C, D), and each of these nodes is assigned to one of the processors. At this point, the processor doing the subtree at node B and C has fewer nodes than does the other processor. Due to this imbalance workload, these two processors are idle and bring the lower efficiency.

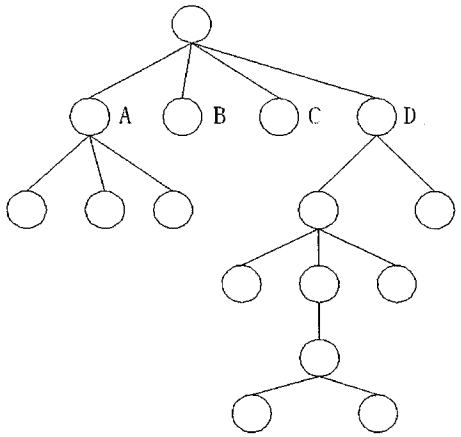


Figure 2.10 The imbalance resulting from static partitioning

On the other hand, in the dynamic situation, the subtrees are dynamically distributed. Consider Figure 2.10 again. The two processors partition the tree. Assume that nodes A and B are assigned to the two processors. In this case when the processor searching the subtree rooted at node B runs out of work, it requests work from the other processor. Although it will result in communication overhead, it still decreases load imbalance. There is no idle processor during the execution. In dynamic load balancing, some important issues should be considered as follows [KR87].

1. Work splitting strategies: it is ideal to send half of the queue in order that the size of search space is the same. This split is called half-split. But it is difficult to determine the amount of work from an unexpanded node in the queue. The alternatives near the bottom of the queue or near the top of the queue tend to have either larger trees or small trees. To avoid sending very small amounts of work, nodes beyond a queue depth are not sent away. This depth is called cutoff depth. Therefore, some strategies are (1) Sending nodes near the bottom of the queue (2) Sending nodes near the cutoff depth (3) Sending half the nodes between the bottom of the queue and the cutoff depth. All of these depend on the nature of search space [GGK+03].

2. Load balancing schemes: Typically, some schemes are often used such as asynchronous round robin, global round robin, the random polling and manager/worker.

Asynchronous round robin allows each processor to have a label and maintain a target of requesting work. Initial value of target is (label +1) mod p. When a processor becomes idle, increment target by 1 (mod p) and send a request for a job to processor number target. Global round robin has a global target maintained by one processor (e.g., P_0). When a processor becomes idle, it asks P_0 for a donor. Random polling means when a processor needs some work, it randomly picks a donor which sends work to others [KGR94].

2.7 State-copying and State-recomputation

Search in constraint programming is a time-consuming task. The searching can be speeded up by exploring subtrees of a search tree in parallel. However, the problem is how to efficiently communicate the state of the search from one processor to another. Many approaches have been developed. There are essentially two ways of setting up the computation state [MS94]: state-copying and state-recomputation.

State-copying has been one of the most successful approaches for solving distributed environment problems and has been used as basis for more complex parallel systems. In a state-copying system, several processors explore different alternatives in the search tree independently. When a processor P_i exhausts its branch and wants to take work with another processor P_j, the current state has to be set up. It then copies the entire state of P_j and backtracks to the choice point in order to undo all the conditional bindings. A processor starts working on the current state without recomputation. Implementation of this approach is not too difficult because state-copying is independent of operations and only concerned with data structures. Unfortunately, state-copying has a little communication overhead.

Instead of the state-copying, the processor can re-compute it from the initial state. The State-recomputation method is motivated by the need for reducing communication

among multiple processors [Shap89]. The State-recomputation method can be achieved by using an oracle such as by keeping track during the original execution and by using this recorded information (*oracle*) to guide the recomputation. Note that the oracle is different from the Oracle database management system. An oracle is just a set of integers. When a processor is assigned an oracle, it tracks the given oracle and takes the corresponding choices. A striking advantage of the recomputation is that a worker communicates by exchanging simple data and the worker does not have to share any state.

The parallel CSP algorithm using State-recomputation mechanism is similar to the algorithm using State-copying mechanism. The only difference is the message. One is an oracle and the other is a state. As compared above, since the size and volume of oracle messages are smaller than queues, and since the workers do not share any state, the recomputation scheme has the potential to overcome the communication bandwidth problem. On the other hand, the copying scheme can be cheaper than recomputation for certain kinds of programs because it has larger messages but less processing.

2.8 SHARCNET

SHARCNET stands for Shared Hierarchical Academic Research Computing Network [Shar01]. It is a world leading computational facility that enables the highest quality of research in critical areas of science, engineering and business, and provides a research platform for studying and implementing HPC. SHARCNET is a multi-institutional high performance computing network and was formally established in 2001. It consists of eleven geographically-distributed HPC clusters at academic institutions across Southern Ontario. The consortium is led by the University of Western Ontario, and it includes the University of McMaster, Guelph, Windsor, Wilfrid Laurier, Waterloo, Brock, Ontario Institute of Technology and York, and Fanshawe and Sheridan colleges.

Figure 2.11 is the community of SHARCNET. It is supported by the Canada Foundation for Innovation, Ontario Innovation Trust, Ontario Research and Development Challenge Fund, and its institutional members and partners which include Hewlett Packard, Platform Computing, Bell Canada, Nortel Networks and Quadrics Ltd.

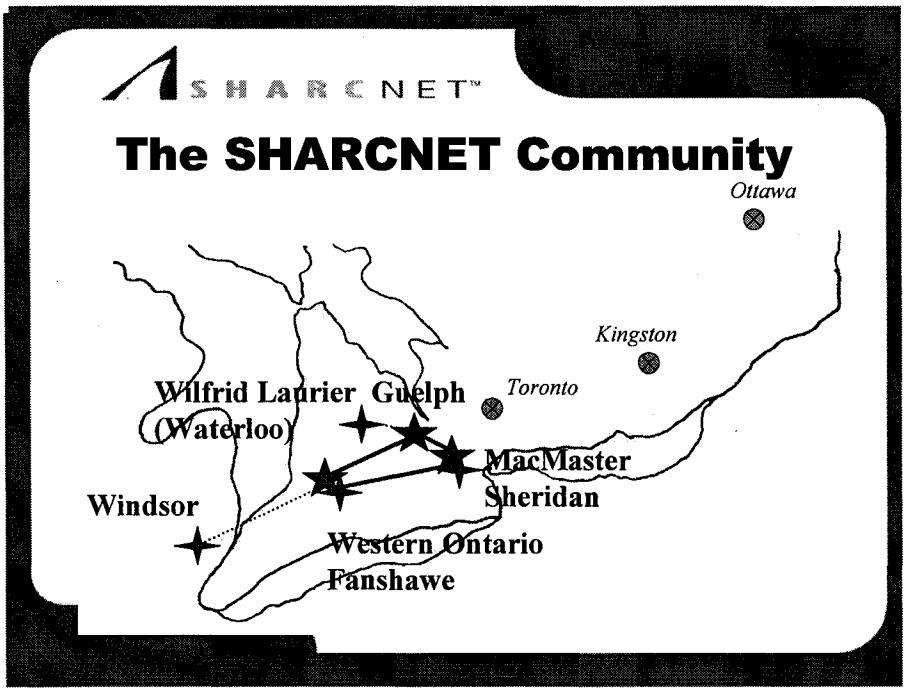


Figure 2.11 The SHARCNET Community

SHARCNET provides scalable computational resources through a hierarchy of processing capability of over 400 HP/Compaq Alpha processors and large symmetric multiprocessor computers. It also provides highly qualified system administrators and high performance computing consultants that ensure the best possible use of time and resources for the SHARCNET community.

2.9 Conclusions

In this chapter we provided a brief introduction to the constraint satisfaction problem and different solving techniques. Also we introduced some basic knowledge of parallel computing. In Chapter 3 we are going to discuss two approaches in detail.

Chapter 3 Description of Approaches

In the forward checking algorithm, a tree structure can be searched for solutions with the sequential constraint directed search. In order to speed up the search work, the processor can be distributed by assigning different branches to different processors. The problem is how to communicate the state of the search from one processor to another. In this chapter we present two mechanisms: state-copying and state-recomputation with forward checking strategy. Both are parallel versions. They will be run on SHARCNET using the message passing interface (MPI), which is a library used to communicate among processors. We then give an example to show how these methods can be applied.

3.1 Basic Ideas

In these parallel versions, the key is how to communicate with each other. A copying based method sends a message which copies the state from one processor to another. In this case, there is a high communication cost. A recomputation based method sends a message containing an oracle which is search path information. In this case, the option must re-compute the state by oracle. This involves many small messages but more processing.

Both methods involve multiple processors and introduce manager/workers architecture. The implementation of both methods comprises three main parts: the manager procedure on manager processor, the worker procedure on worker processors,

and the routine search, forward checking algorithm on worker processors. We designate one processor as manager whose task is to schedule the jobs among the other processors and control the process. It receives the work request message and work message from all the workers. We also designate other processors as workers whose task is to do the routine search (forward checking algorithm) and report the state to the manager if needed. The manager and workers communicate with each other via the message passing interface. Figure 3.1 illustrates the architecture of both methods.

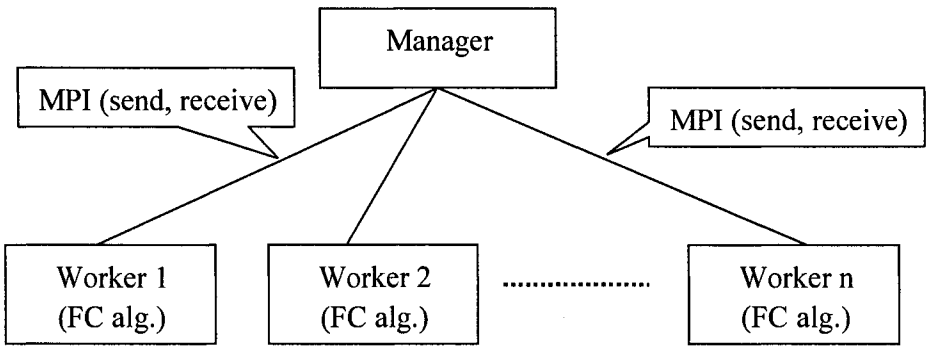


Figure 3.1 Architecture of both methods

3.2 Forward Checking Algorithm in State-copying

In the state-copying based forward checking algorithm, workers build a state by copying its current state. When a worker runs out of its job and become idle, it will send a request to the manger. If the manager asks for more jobs from the donator (busy worker), the donator sends the job to the manager, and then the manager sends it to the request worker. When the worker receives a job message from the manager, it starts to compute using this copying state. When a worker has done its job, it informs the manager and needs more jobs until all workers finish. The following figure 3.2 and figure 3.3 are the flowcharts of a state-copying based algorithm.

Manager flow chart

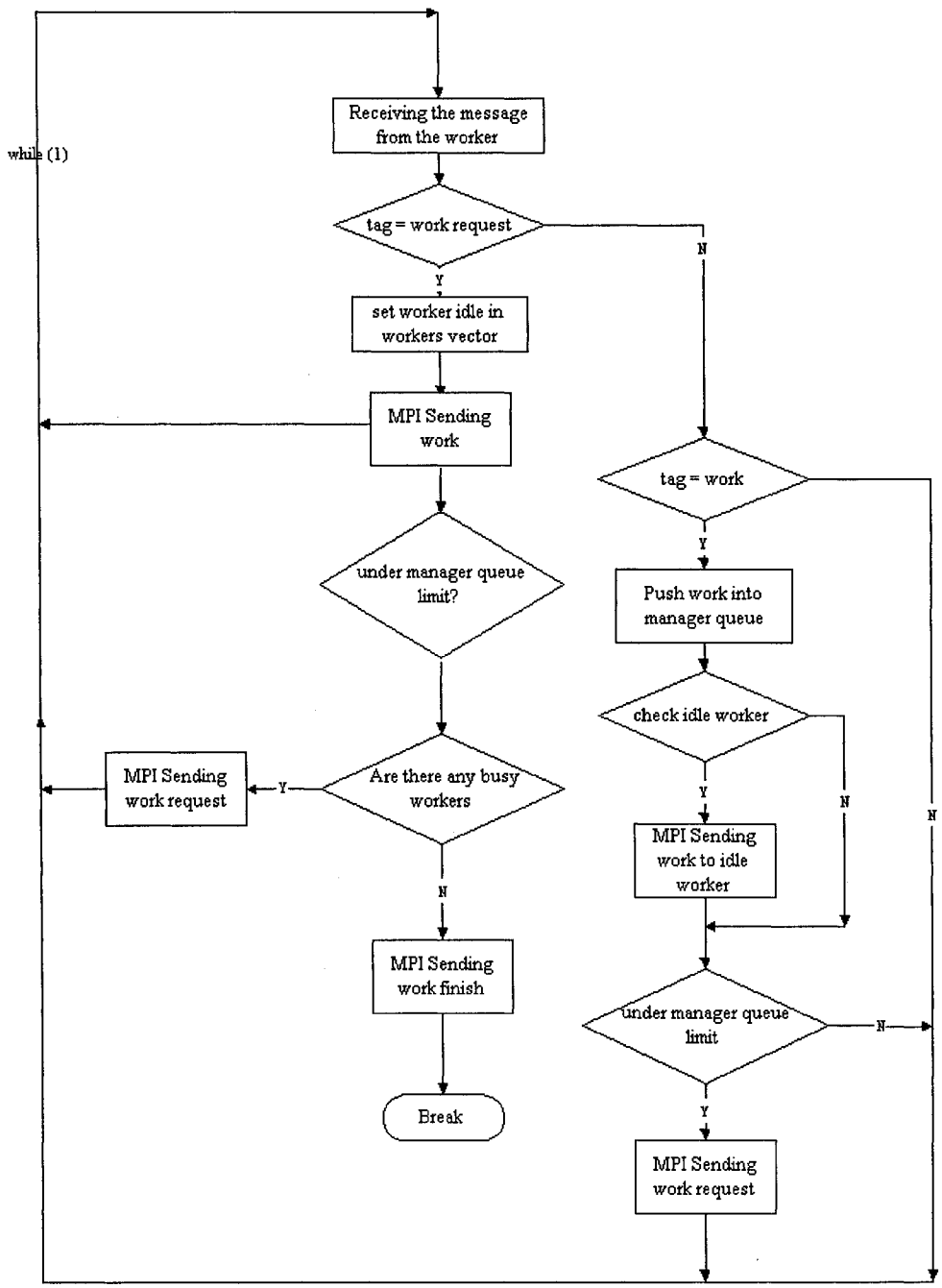


Figure 3.2 Manager flow chart of state-copying algorithm

Worker flow chart

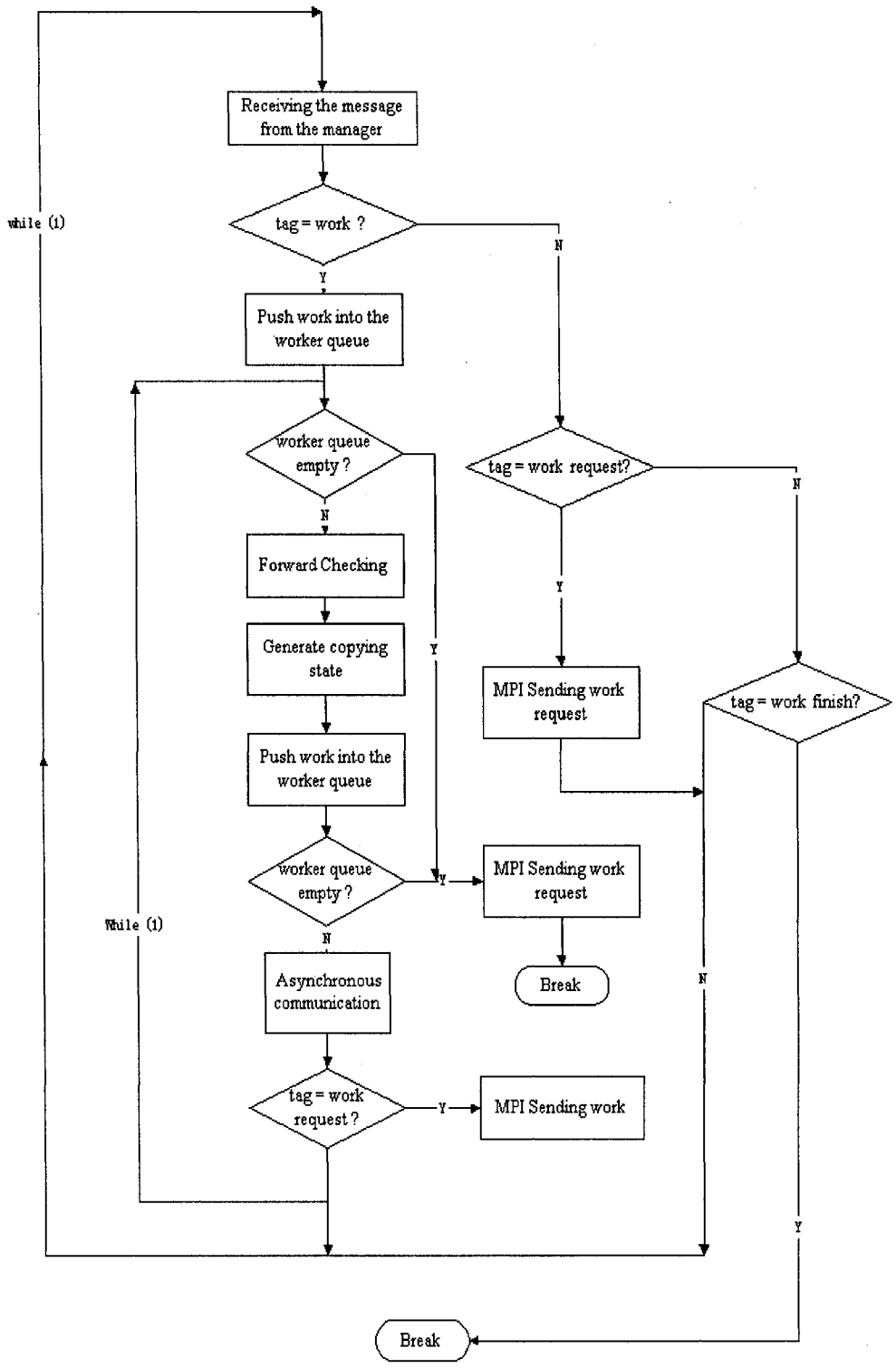


Figure 3.3 Worker flow chart of state-copying algorithm

3.3 A Simple Example of State-copying based Algorithm

We give a simple example to illustrate how this algorithm can be applied. The following non-binary CSP has 4 variables X_1, X_2, X_3, X_4 with domain size {1,2,3}, and 3 constraints with tightness 0.2, 0.5 and 0.5. See the table 3.1 below.

X_1	Domain size: {1, 2, 3}		
X_2	Domain size: {1, 2, 3}		
X_3	Domain size: {1, 2, 3}		
X_4	Domain size: {1, 2, 3}		
C_1	$\{X_1, X_2, X_3\}$	Tightness: 0.2	{1,1,1},{2,2,3},{2,3,3},{3,1,1},{3,3,3}
C_2	$\{X_1, X_4\}$	Tightness: 0.5	{1,1},{1,2},{1,3},{3,3}
C_3	$\{X_2,X_3\}$	Tightness:0.5	{1,1},{2,2}{3,1},{3,3}

Table 3.1 An example of non-binary CSP

Suppose we submit the job and require 3 processors (P_0, P_1, P_2). One (P_0) is taken as a manager and the others (P_1,P_2)are taken as workers. In both programs, the first part of program is to transform the non-binary CSP to a binary CSP. After the transformation, the new binary CSP variables are: V_1:$\{X_1, X_2, X_3\}$, V_2:$\{X_1, X_4\}$ and V_3:$\{X_2,X_3\}$, the new binary CSP domain size are: $D(V_1)$={{1,1,1},{2,2,3},{2,3,3},{3,1,1},{3,3,3}}, $D(V_2)$={{1,1},{1,2},{1,3},{3,3}} and $D(V_3)$={{1,1},{2,2}{3,1},{3,3}}. The second part of program is to search job in parallel. Based on Figure 3.2 and Figure 3.3, they work as follows:

1. For each value of V_1, the program sets up the current states and pushes them to the manager queue of the jobs. Then the manager starts to distribute the jobs to workers.
2. It also sets up a worker vector to check the worker state: idle or busy, and sets up manager queue limitation, for example, the limitation is at least 2 jobs in the manager queue in this case.

3. The manager sends jobs to 2 workers. One job is {1,1,1} to P_1, the other job is {2,2,3} to P_2. The job message is state copying.

4. When P_1 receives the job {1,1,1}, it sets up the state. P_1 starts to do forward checking algorithm. When P_2 receives the job {2,2,3}, it sets up the state. P_2 starts to do forward checking algorithm. Both processors work at the same time.

5. According to the new constraints of the binary CSP for P_1, V_2 removes {3,3} and V_3 removes {2,2},{3,1}, and {3,3}. For P_2, V_2 removes all values and that domain is empty. So P_2 become idle. P_2 sends a request to manager for a job.

6. For P_1, it creates the current state and pushes V_1={1,1,1},V_2={1,1}, V_1={1,1,1},V_2={1,2} and V_1={1,1,1}, V_2={1,3} into the worker queue.

7. For P_1, it picks V_2={1,1}and check the constraints. So it gets V_3={1,1}. It outputs the solution: {1,1,1},{1,1}and {1,1}. P_1 continues to do a job, it picks up V_1={1,1,1},V_2={1,2}.

8. When the manager receives the job request from P_2, it sends the job V_1={2,3,3} and checks the manager queues. If it reaches the limitation, it sends the job request to a busy worker. In this case, there is a non blocking mechanism. P_1 receives it and sends back {1,1,1}{1,3} while P_1 is doing its job. P_1 picks V_3={1,1} and gets another solution: {1,1,1},{1,2} and {1,1}. P_1 makes a job request to the manager.

9. When P_2 gets {1,1,1}{1,3}, it finds the solution {1,1,1}{1,3} and {1,1}. When P_2 gets V_1={2,3,3} and does forward checking, it violates the constraints and the domain is empty. It sends a job request again from the manager.

10. There are idle workers, so the manager sends {3,1,1} and {3,3,3} to them. P_1 gets V_1={3,1,1} and removes V_2 = {1,1},{1,2} and {1,3} and V_3={2,2},{3,1},{3,3}. P_2 gets V_1={3,3,3} and removes V_2={1,1},{1,2} and {1,3} and V_3={1,1},{2,2},{3,1}.

11. For P_1, it picks V_2={3,3} and then picks V_3={1,1}. The solution for P_1 is {3,1,1},{3,3} and {1,1}. For P_2, it picks V_2={3,3} and then picks V_3={3,3}. The solution for P_2 is {3,3,3}, {3,3} and {3,3}.

12. P_1 and P_2 send idle states to manager. The manager checks the worker vector and finds the jobs are done. So it sends a finish message to all workers.

13. Finally, we get all the solutions. They are: {1,1,1},{1,1},{1,1}; {1,1,1},{1,2},{1,1}; {1,1,1},{1,3},{1,1}; {3,1,1},{3,3}, {1,1}; {3,3,3},{3,3},{3,3}.

In this scenario all the communication messages are about state copying, which means states are sent among processors. When the processor gets the message, it does not need to re-compute the state but has high communication costs.

3.4 Forward Checking Algorithm in State-recomputation

In the state-recomputation based forward checking algorithm, workers create an oracle associated with current search path. When a worker runs out of its job and becomes idle, it will send a request to the manger. If the manger asks for more jobs from the donator (busy worker), the donator sends the job to manager, and then the manager sends it to the request worker. When the worker receives a job message from the manager, it starts to compute using this copying state. When a worker has done its job, it informs the manager and needs more jobs until all workers finish. The following figure 3.4 and figure 3.5 are the flowcharts of state-copying based algorithm.

Manager flow chart

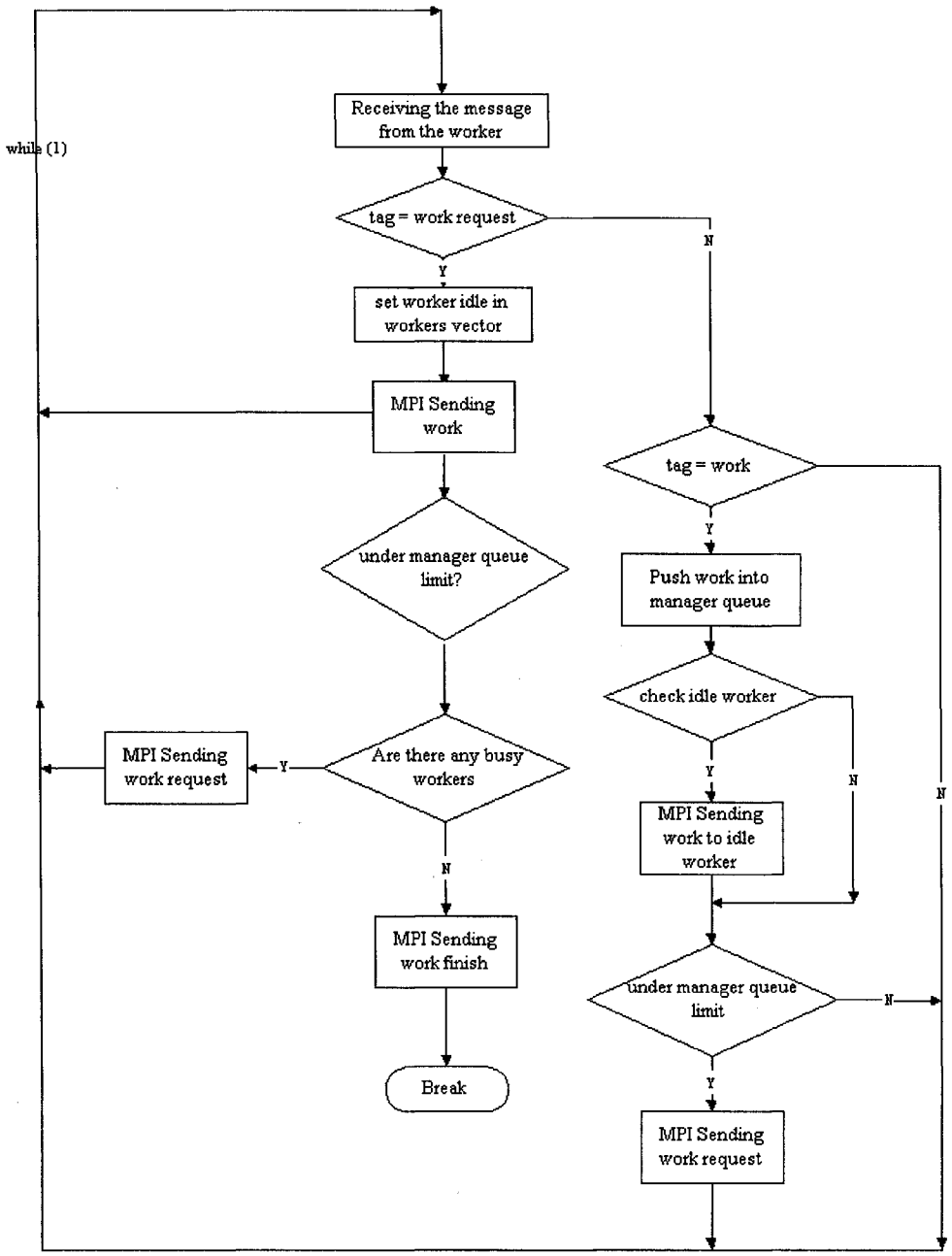


Figure 3.4 Manager flow chart of state-recomputation algorithm

Worker flow chart

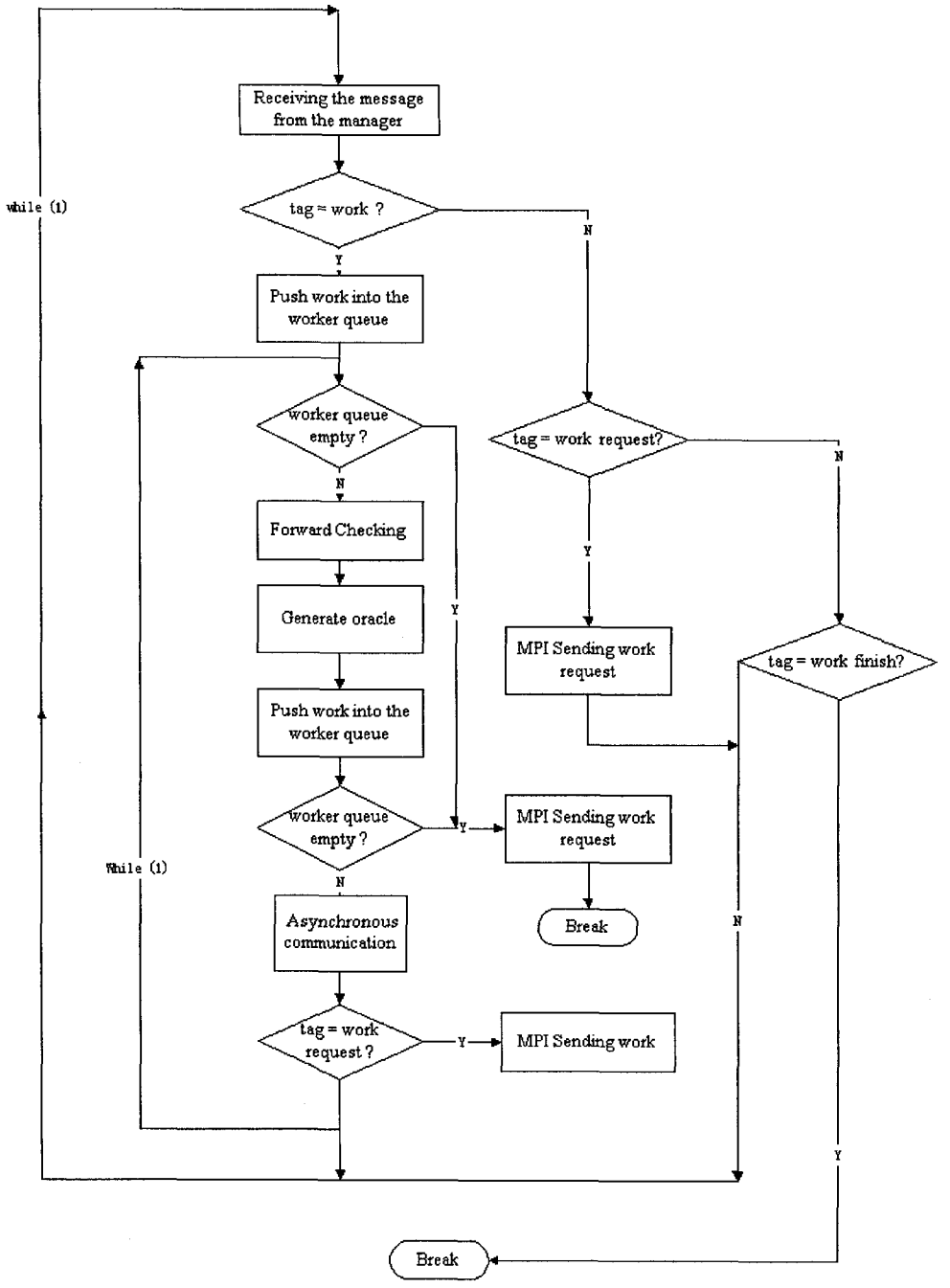


Figure 3.5 Worker flow chart of state-recomputation algorithm

3.5 A Simple Example of State-recomputation based Algorithm

We give a simple example to illustrate how this algorithm can be applied. We are given the non-binary CSP which is the same as the above with 4 variables X_1, X_2, X_3, X_4 with domain size {1,2,3}, and it has 3 constraints with tightness 0.2, 0.5 and 0.5. Suppose we submit this job and require 3 processors (P_0, P_1, P_2). One (P_0) is taken as a manager and the others (P_1,P_2) are taken as workers. The first part of program is to transform non-binary CSP to binary CSP. After the transforming, the new binary CSP variables are: V_1:{X_1, X_2, X_3}, V_2:{X_1, X_4} and V_3:{X_2,X_3}, the new binary CSP domain size are: $D(V_1)$={1,1,1},{2,2,3},{2,3,3},{3,1,1},{3,3,3}; $D(V_2)$={1,1},{1,2},{1,3},{3,3} and $D(V_3)$={1,1},{2,2}{3,1},{3,3}. The second part of the program is to search job in parallel. Based on Figure 3.4 and Figure 3.5, they work as follows:

1. For each value of V_1, the program sets up the oracles and pushes them to the manager queue of the jobs. Then the manager starts to distribute the jobs to workers.
2. It also sets up a worker vector to check the worker state: idle or busy, and sets up manager queue limitation, for example, the limitation is at least 2 jobs in the manager queue in this case.
3. The manager sends jobs to 2 workers. One job is oracle (1) to P_1, the other job is oracle (2) to P_2. The job message is oracle.
4. When P_1 receives the job, oracle (1), it re-computes the state and starts to do the forward checking algorithm. When P_2 receives the job, oracle (2), it re-computes the state and starts to do the forward checking algorithm. Both processors work at the same time.
5. According to the new constraints of the binary CSP, For P_1,V_2 removes {3,3} and V_3 removes {2,2},{3,1}, and {3,3}. For P_2, V_2 removes all values and that domain is empty. So P_2 become idle. P_2 sends a request to the manager for a job.
6. For P_1, it creates the current state and pushes oracle (1,2) and oracle (1,3) into the worker queue.

7. For P_1, it picks V_2={1,1}and check the constraints. So it gets V_3={1,1}. It outputs the solution: oracle (1,1,1). P_1 continues to do a job, it picks up oracle (1,2).

8. When the manager receives the job request from P_2, it sends the job, oracle (3) and checks the manager queues. If it reaches the limitation, it sends the job request to a busy worker. In this case, there is a non blocking mechanism. P_1 receives it and sends back, oracle (1,3) while P_1 is doing its job. P_1 picks V_3={1,1} and gets another solution: oracle (1,2,1). P_1 makes a job request to the manager.

9. When P_2 gets oracle (1,3), it finds the solution oracle (1,3,1). When P_2 gets oracle (3) and does forward checking, it violates the constraints and the domain is empty. It sends a job request again to the manager.

10. There are idle workers, so the manager sends oracle (4) and oracle (5) to them. P_1 gets oracle (4) and removes V_2 = {1,1},{1,2} and {1,3} and V_3={2,2},{3,1},{3,3}. P_2 gets oracle (5) and removes V_2={1,1},{1,2} and {1,3} and V_3={1,1},{2,2},{3,1}.

11. For P_1, it picks V_2={3,3} and then picks V_3={1,1}. The solution on P_1 is oracle (4,4,1). For P_2, it picks V_2={3,3} and then picks V_3={3,3}. The solution for P_2 is oracle (5,4,4).

12. P_1 and P_2 send idle states to manager. The manager checks the worker vector and finds the jobs are done. So it sends a finish message to all workers.

13. Finally, we get all the solutions. They are: (1,1,1); (1,2,1); (1,3,1); (4,4,1); (5,4,4).

In this scenario all the communication messages are about oracles which are search paths. When the processor gets the message, it needs to re-compute the state. Although it involves many smaller messages, it has more computing.

3.6 Conclusions

In this chapter we discussed the details of both algorithms. The big difference between them is the communication overhead versus recomputation overhead. In the next

chapter, we will evaluate the performance of both algorithms and determine which model is better.

Chapter 4 Experiment and Evaluation

In chapter 3 we presented two approaches in detail. In this experimental study, we investigate the behaviour of state-copying based and state-recomputation based forward checking strategy on dual representation for many different non-binary CSPs. We implement them on the SHARCNET environment. The programming language is C++ STL with MPI library. The objective of this experiment is to evaluate the performance of both algorithms and compare these two approaches to find which one has higher efficiency and what sorts of circumstances would be favorable for them.

4.1 Experimental Input and Output Design

We randomly generate non-binary CSPs based on a four parameter model from the standard four parameter binary CSP model [Smit94]. The four-parameter model is as follows: Number of variables (n); size of individual domains (m); probability of tuple exclusion (P_t); probability of constraint inclusion (P_c); where P_t is the tightness of the constraint and P_c is the constraint density [Naga01]. For example, when the fixed arity is 3, variables n = 10 and density $P_c = 0.04$, we generate a CSP with (10*9*8/3*2) * 0.04 = 4 constraints. In our experiments, we generate 3 classes to analyze the performance of approaches. The classes are given as {n, a, d, P_c} where n is the number of variables, a is the arity, d is domain size and P_c is the constraint density. The 3 classes are as follows:

- Class Ⅰ: {10, 3, 10, 0.04}
- Class Ⅱ: {10, 3, 15, 0.04}

- Class III: {15, 4, 9, 0.005}

Each class includes 3 instances which mean each runs on different processors (3 processors, 6 processors, 9 processors) and that each instance has 16 problems where the constraint tightness changes from low value to high value(16 steps). For each value of tightness we generate 10 problems on both approaches, and all of results are analyzed by the t-test. For each problem, there is a plain text file as an input file and the program generates the output file which evaluates the performance. The input file includes:

- The number of variables
- The domain size related to each variable
- The number of constraints
- Each constraint with given arity
- Each constraint with tightness

Each program will read the input file, transform the non-binary CSP to a binary CSP by the dual graph method and parallel computing among several processors, and then generate the output file. In order to obtain the performance analysis, the output file should include as follows:

- The solution of the CSP
- How long does the program take to get the solution? It includes two parts: one is the elapsed time of transforming non-binary CSP to binary CSP, the other is the elapsed time of searching solutions.
- How much time is spent in recomputation?
- How many message sizes are communicated between processors?

4.2 Experimental Framework

We investigate and evaluate the behavior of state-copying and state-recomputation approaches on different processors over several problem instances during the experiments. It is important to study the performance of the parallel program and to examine the benefits from parallelism. The parallel metrics should be used based on the performance analysis.

4.2.1 Execution Time

The parallel execution time is the time that begins from the moment a parallel program starts to the moment the last processing unit finishes execution. We denote the parallel execution time by T_P.

4.2.2 Speedup

We are interested in knowing how much performance gain is achieved by parallelizing a non-binary CSP with the forward checking algorithm. Speedup is the measurement that captures the relative benefits. It is defined by $S = T_S / T_P$. The symbol S is the speedup, T_S is the time the fastest serial program takes to run, and T_P is the time it takes to run the same problem with N processors. T_S is supposed to be the run time of the best possible sequential algorithm, but in general, the best possible algorithm is unknowable. The parallel algorithm run on one processor is often used in the place of T_S. T_P should be smaller than T_S because the parallel program runs faster than the sequential algorithm. The larger the value of S, the better the performance is. In this experiment we intend to compare the speedup of these two approaches.

4.2.3 Efficiency

Only an ideal parallel program can obtain a speedup equal to the number of processors. However, in practice, ideal behavior is not achieved because the processors cannot devote 100% of their time to the computation. Efficiency is a measurement that a processor is employed. It is the ratio of speedup to the number of processors. It is mathematically defined by E = S / N, where S is speedup and N is the number of processors. In an ideal parallel program, speedup is equal to N and efficiency is equal to one, but in practice, speedup is less than N and efficiency is between zero and one. In this experiment, we will examine the efficiency of these two approaches.

4.2.4 Communication and Recomputation

The main goals of this evaluation are to compare the merits of state copying versus recomputation. The approach of state copying is characterized by large amount of communication with relatively little calculation while the approach of state recomputation is characterized by intensive computation. Because the parallel execution runtime consists of computation and communication time, the trade-off between these two have a significant and serious impact. In order to obtain rules and draw conclusions, the analysis and study about trade-off must be conducted.

4.3 Experimental Analysis Approaches

We conduct a statistical analysis method called the t-test to compare the two approaches in order to support our conclusion. The traditional method to obtain a performance analysis is to run the program many times on the same problem instance and then to calculate the average value. For example, we run the state copying based program on the non-binary CSP (variable (4), domain size (10), and constraints (4) with given arity, tightness) for 20 times or more, and we can get the mean of execution time

by dividing the sum of execution times by 20. On the other side, we run the state recomputation based program on the same non-binary CSP and get the mean execution time. Comparing the sample means of two groups is a common way of conducting an experimental study. However, sample results may lead to incorrect conclusions about the two populations. For example, if the sample mean of group 1 is greater than the sample mean of group 2 with a difference such as 0.001, could we say the population performance time of group 1 is greater than the population performance time of group 2? How could we verify? In this experimental study we utilize the t-test to solve the problem above. The t-test can be applied to a relatively small number of cases. It is generally used to evaluate statistical differences for samples of 30 or less. When the sample size is larger than 30, the test for assessing the difference of the populations means between two groups becomes a Z-test. The t-test is used to evaluate whether the population means of two groups differ. The first step is to establish the specific hypotheses. The t-test uses two hypotheses, a null hypothesis and an alternative hypothesis.

1. H_0: $\mu_A = \mu_B$

The null hypothesis declares that the difference between the population means of two groups is zero. For example, the population mean execution time of the state-copying based program is the same as the population mean execution time of the state-recomputation based program.

2. H_1: $\mu_A > \mu_B$ (or $\mu_A < \mu_B$)

The alternative hypothesis states that the difference between the population means of two groups is not zero. For example, the population mean of group A is greater than (or less) than the population mean of group B.

The calculation of the T-test statistic requires three additional components:

1. The sample average value of both groups. Statistically we represent these as: $\overline{X}_A$ and $\overline{X}_B$.

2. The sample standard deviation of both groups. Statistically, we represent these as: S_A and S_B.

3. The number of observations in both groups. Statistically, we represent these as n_A and n_B.

With these components, the formula for the t-value is as follows:

T_value = (difference between sample means) / (standard deviation)

$$= (\overline{X_A} - \overline{X_B}) \;/\; S(\overline{X_A} - \overline{X_B})$$

$$= (\overline{X_A} - \overline{X_B}) \;/\; \mathrm{SQRT}(S_A^2/n_A^2 + S_B^2/n_B^2)$$

We need a risk level which is referred to as alpha level. The alpha level is conventionally set at 0.05 which means that five times out of a hundred you will detect a significant population difference when there is none present. We recommend that a confidence interval also be calculated. A confidence interval indicates how much uncertainty there is. In practice, a 95% confidence level is commonly used. For a 95% confidence interval, if many samples are gathered and the confidence interval is computed for each sample, then about 95% of these intervals would contain the true difference of the means. Besides that, we also need the degrees of freedom (df) for the t-test. For equal variances, the degrees of freedom (df) is the sum of the number of rounds of running for both groups minus 2. For example, we run the group A and group B 10 times, and get the df for the t-test as 10+10-2=18. For unequal variances, the degrees of freedom (df) can be obtained. We assume that before we do experiments, we have a hypothesis that the population mean of one group is greater than or less than the other mean. This means the t-test is in the one-tailed t-test. If the alternative hypothesis says that the population means are not equal, then we would need the two-tailed t-test. In our experiment study, we focus on whether the mean of execution time for state-copying is greater than or less than the mean of execution time of state-recomputation. Therefore our test is the one-tailed t-test.

Having calculated the t-statistic, we compare the t-value with a standard table of t-values (T_criticalvalue, T_cv). This table is called T-table. The T-table is used to determine whether the t-statistic reaches the threshold of statistical significance. In this experiment, given $\alpha = 0.05$, we look up the T-table and find out the T_cv = 1.717144.

We compare the T_cv with T-value. If T_value > T_cv, we draw the conclusion that the difference between the means of the two groups is statistically significant, and if T_value is positive, we can say that the population mean execution time of group A is greater than the population mean execution time of group B. If T_value is negative, we can say that the population mean execution time of group B is greater than the population mean execution time of group A. If T_value < T_cv, we conclude that the two population means times are not significantly difference.

4.4 Results

Some notations are described below.

Average Elapsed Time of Recomputation: **ET-R** (in seconds)

Average Elapsed Time of Copy: **ET-C** (in seconds)

Recomputation Time: **RT**; **Ratio** is the recomputation time to the elapsed time

Transferring message size of recomputation: **MR** (bytes);

Transferring message size of copying: **MC** (bytes);

Speedup of recomputation: **SpR**; Speedup of copying: **SpC**;

Efficiency of recomputation: **EffR**; Efficiency of copying: **EffC**;

4.4.1 Experimental Results of Class Ⅰ on 3 Processors

Table 4.1 contains the elapsed time of both approaches of Class Ⅰ on 3 processors. The steps represent the tightness from 1.0 to 10.0. We use the t-test to compare the two groups. As discussed above, we calculate the T-value and compare it with T_cv = 1.717. The elapsed time is measured in seconds. If |T_value| > T_cv, the difference between the means of the two groups is statistically significant. Also, if T_value is positive (or negative), we can say that the population mean execution time of group A is greater than (or less than), respectively, the population mean execution time of group B. If

|T_value|< T_cv, we conclude that the two population means times are not significantly different. In Table 4.1 we also provide the confidence interval.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
1.0	0.100	0.125	-8.293	-0.031	-0.019	Recom
1.5	0.196	0.281	-26.593	-0.092	-0.079	Recom
1.8	0.451	0.551	-3.910	-0.154	-0.046	Recom
2.0	0.753	0.809	-4.863	-0.080	-0.032	Recom
2.5	1.269	1.400	-7.015	-0.170	-0.092	Recom
3.0	2.053	2.306	-8.946	-0.313	-0.194	Recom
3.2	2.468	2.749	-9.608	-0.343	-0.220	Recom
3.4	3.996	4.150	-3.426	-0.248	-0.060	Recom
3.6	**5.871**	**5.614**	**4.744**	**0.143**	**0.371**	**Copying**
3.8	7.943	7.453	4.551	0.264	0.716	Copying
4.0	9.996	9.261	3.972	0.346	1.123	Copying
4.5	18.069	15.720	13.027	1.970	2.728	Copying
5.0	25.359	22.186	8.096	2.349	3.996	Copying
6.0	65.059	55.477	11.452	7.824	11.339	Copying
8.0	198.955	161.067	18.972	33.692	42.084	Copying
10.0	707.223	519.135	35.257	176.879	199.295	Copying

Table 4.1 Elapsed time of both approaches of Class I on 3 processors

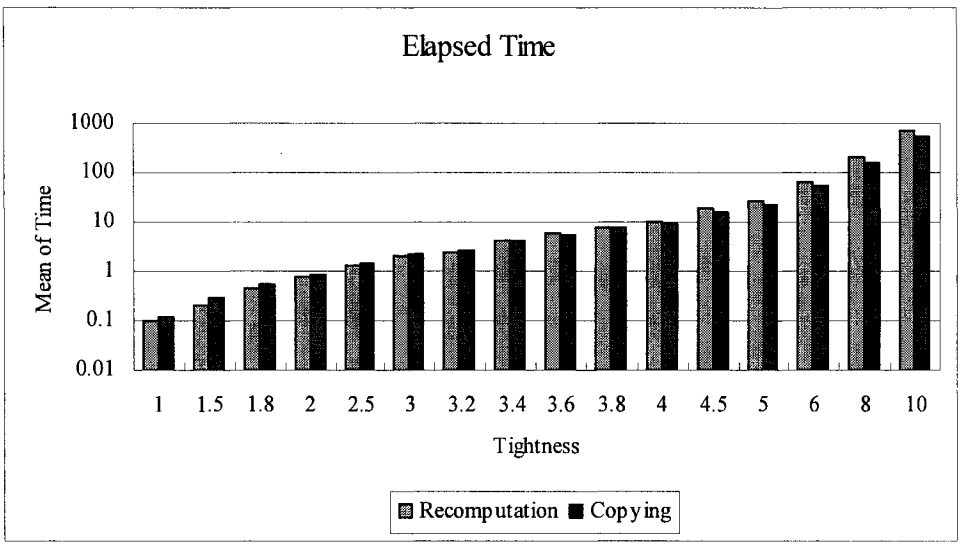


Figure 4.1 Mean of time of Class I for recomputation and copying on 3 processors

We present Figure 4.1 for the equivalent Table 4.1. The X axis measures tightness and the Y axis is based on a logarithmic scale. In Figure 4.1 we observe the

state-recomputation elapsed time is faster than state-copying when the tightness is less than 3.6. When the tightness is equal to or greater than 3.6, the state-copying elapsed time is faster than the state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
1.0	0.018	0.186	10	440	1.646	1.316	0.549	0.439
1.5	0.037	0.188	15	960	1.677	1.168	0.559	0.389
1.8	0.130	0.288	24	1520	1.752	1.434	0.584	0.478
2.0	0.238	0.316	28	1730	1.956	1.822	0.652	0.607
2.5	0.438	0.345	36	2621	1.947	1.765	0.649	0.588
3.0	0.716	0.349	30	3720	1.998	1.779	0.666	0.593
3.2	0.913	0.370	32	4224	1.991	1.787	0.664	0.596
3.4	1.507	0.377	34	4760	1.875	1.806	0.625	0.602
3.6	**2.851**	**0.486**	**47**	**5328**	**1.894**	**1.981**	**0.631**	**0.660**
3.8	4.048	0.510	38	5928	1.775	1.892	0.592	0.631
4.0	6.430	0.643	44	6822	1.856	2.004	0.619	0.668
4.5	9.755	0.540	46	8979	1.652	1.898	0.551	0.633
5.0	12.638	0.498	52	10730	1.747	1.996	0.582	0.665
6.0	34.950	0.537	74	15177	1.635	1.917	0.545	0.639
8.0	112.450	0.565	102	28577	1.603	1.980	0.534	0.660
10.0	426.514	0.603	121	42258	1.497	2.039	0.499	0.680

Table 4.2 Parallel metrics of both approaches of Class I on 3 processors

The main parallel execution overhead of the state-recomputation model is due to recomputation and communication, whereas the overhead of the state-copying model is mainly due to communication. Table 4.2 shows the ratio of recomputation time to the elapsed time. In this case when the tightness is greater than 3.6 and the ratio of recomputation time to elapsed time is above 48%, then the overhead of state-recomputation model is much higher and the run time of state-copying model is faster than that of the state-recomputation model. Table 4.2 also provides a rough measurement of the communication overhead, which is obtained by examining the total message sizes sent out by the manager and workers. When the tightness is less than 3.6, the message size of recomputation is quite smaller and recomputation ratio is lower, thus the recomputation performance is better than the copying performance. Here we also illustrate the speedup and efficiency. The speedups are expected to be greater with

the increase of processors. Both approaches also have similar characteristics with respect to the speedup and efficiency.

4.4.2 Experimental Results of Class I on 6 Processors

Table 4.3 contains the elapsed time of both approaches of Class I on 6 processors. When we use the t-test to compare the two groups, we observe that the |T_value| < T_cv when the tightness is between 2.0 and 3.4, inclusive. We conclude that the two population means times are not significantly different. In Table 4.3 we also provide the confidence interval.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
1.0	0.078	0.085	-1.090	-0.021	0.007	No diff
1.5	0.103	0.142	-10.304	-0.047	-0.031	Recom
1.8	0.241	0.298	-11.430	-0.067	-0.046	Recom
2.0	0.431	0.417	1.236	-0.010	0.038	No diff
2.5	0.723	0.747	-1.261	-0.063	0.016	No diff
3.0	1.006	1.037	-1.253	-0.085	0.021	No diff
3.2	1.300	1.319	-0.439	-0.106	0.069	No diff
3.4	1.811	1.870	-1.662	-0.133	0.016	No diff
3.6	**2.665**	**2.476**	**6.168**	**0.125**	**0.254**	**Copying**
3.8	3.643	3.380	5.545	0.164	0.363	Copying
4.0	4.517	4.099	8.694	0.164	0.363	Copying
4.5	7.643	6.638	10.817	0.810	1.200	Copying
5.0	10.607	9.217	18.116	1.229	1.551	Copying
6.0	27.518	22.643	12.422	4.051	5.700	Copying
8.0	82.169	64.686	16.290	15.228	19.738	Copying
10.0	285.866	215.832	22.954	63.624	76.444	Copying

Table 4.3 Elapsed time of both approaches of Class I on 6 processors

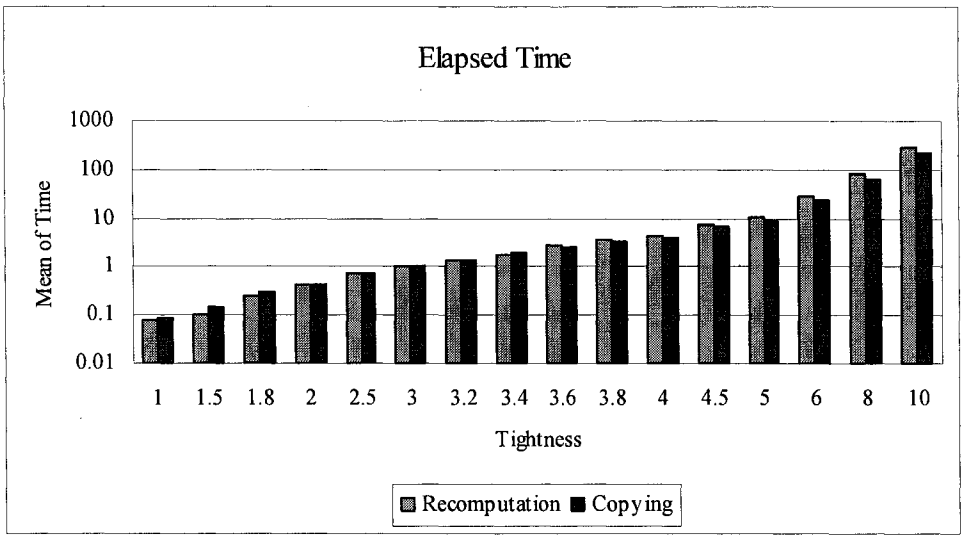


Figure 4.2 Mean of time of Class I for recomputation and copying on 6 processors

We present Figure 4.2 for the equivalent Table 4.3. The X axis represents tightness and the Y axis is based on a logarithmic scale. In Figure 4.2 we observe that the state-recomputation elapsed time is faster than or equal to state-copying when the tightness is less than 3.6. When the tightness is equal to or greater than 3.6, the state-copying elapsed time is faster than state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
1.0	0.010	0.126	18	484	2.112	1.935	0.352	0.323
1.5	0.018	0.177	15	960	3.193	2.316	0.532	0.386
1.8	0.099	0.412	26	1626	3.270	2.647	0.545	0.441
2.0	0.155	0.360	49	2251	3.415	3.530	0.569	0.588
2.5	0.318	0.440	43	3224	3.417	3.308	0.569	0.551
3.0	0.401	0.399	36	3968	4.079	3.955	0.680	0.659
3.2	0.420	0.323	50	4831	3.778	3.725	0.630	0.621
3.4	0.733	0.405	54	5600	4.139	4.009	0.690	0.668
3.6	**1.155**	**0.433**	**81**	**7134**	**4.173**	**4.491**	**0.695**	**0.749**
3.8	1.999	0.549	75	8081	3.869	4.171	0.645	0.695
4.0	2.095	0.464	75	8561	4.109	4.528	0.685	0.755
4.5	4.109	0.538	63	10010	3.905	4.496	0.651	0.749
5.0	5.616	0.529	84	11383	4.176	4.806	0.696	0.801
6.0	15.914	0.578	88	17080	3.866	4.698	0.644	0.783
8.0	44.705	0.544	129	31428	3.881	4.930	0.647	0.822
10.0	178.945	0.626	165	51227	3.703	4.905	0.617	0.817

Table 4.4 Parallel metrics of both approaches of Class I on 6 processors

Table 4.4 shows the ratio of recomputation time to the elapsed time. In this case when the tightness is greater than 3.6, and the ratio of recomputation time to elapsed time is above 43%, then the overhead of state-recomputation model is higher, and the run time of the state-copying model is faster than that of the state-recomputation model. Table 4.4 also shows that when the tightness is less than 3.6, the message size of recomputation is quite smaller and recomputation ratio is lower, thus the recomputation performance is better than or equal to the copying performance. This table also illustrates the speedup and efficiency.

4.4.3 Experimental Results of Class I on 9 Processors

Table 4.5 contains the elapsed time of both approaches of Class I on 9 processors. When we use the t-test to compare the two groups, we observe that the |T_value| < T_cv, when the tightness is between 2.5 and 3.4, inclusive. That means the two population means times are not significantly different. In Table 4.5 we also provide the confidence interval.

Steps	ET-R	ET-C	T-value	CI Lower	CI Upper	Result
1.0	0.080	0.079	0.103	-0.010	0.011	No diff
1.5	0.083	0.109	-6.030	-0.035	-0.017	Recom
1.8	0.199	0.255	-10.665	-0.067	-0.045	Recom
2.0	0.328	0.372	-7.204	-0.056	-0.031	Recom
2.5	0.592	0.599	-0.389	-0.045	0.031	No diff
3.0	0.763	0.781	-0.962	-0.058	0.022	No diff
3.2	1.004	0.987	0.576	-0.045	0.078	No diff
3.4	1.332	1.309	0.839	-0.034	0.079	No diff
3.6	**2.040**	**1.807**	**8.863**	**0.177**	**0.288**	**Copying**
3.8	2.587	2.427	2.946	0.046	0.273	Copying
4.0	3.149	2.990	2.963	0.046	0.271	Copying
4.5	5.251	4.566	13.249	0.576	0.793	Copying
5.0	7.477	6.413	9.072	0.817	1.309	Copying
6.0	17.950	15.423	11.604	2.069	2.985	Copying
8.0	51.917	41.519	30.425	9.680	11.116	Copying
10.0	184.343	138.194	40.739	43.769	48.529	Copying

Table 4.5 Elapsed time of both approaches of Class I on 9 processors

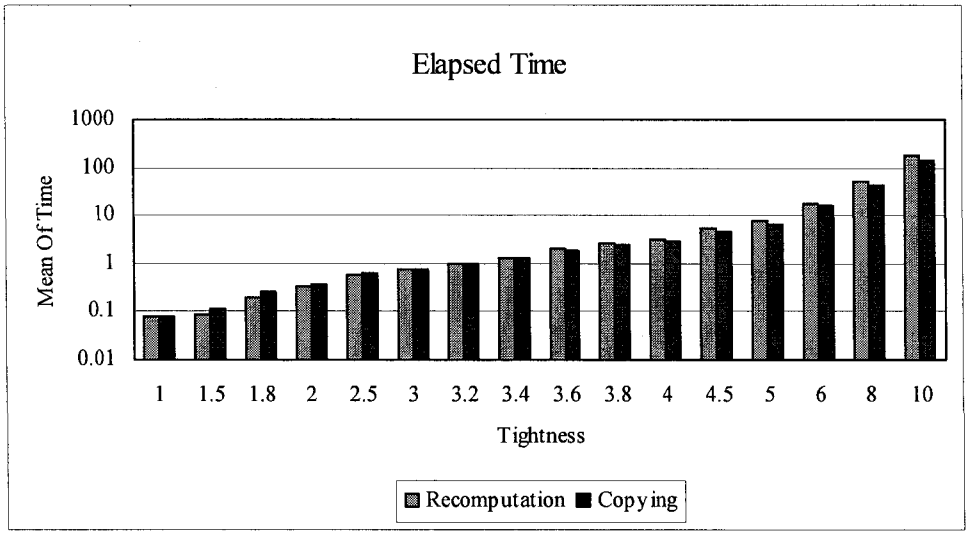


Figure 4.3 Mean of time of Class I for recomputation and copying on 9 processors

Figure 4.3 clearly shows the state-recomputation elapsed time is faster than or equal to state-copying when the tightness is less than 3.6. When the tightness is equal to or greater than 3.6, then the state-copying elapsed time is faster than that of the state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
1.0	0.010	0.121	19	581	2.052	2.065	0.228	0.229
1.5	0.019	0.231	15	960	3.954	3.004	0.439	0.334
1.8	0.047	0.238	33	1702	3.965	3.094	0.441	0.344
2.0	0.112	0.342	42	2539	4.488	3.965	0.499	0.441
2.5	0.213	0.359	48	3245	4.173	4.123	0.464	0.458
3.0	0.314	0.412	55	4166	5.379	5.253	0.598	0.584
3.2	0.406	0.404	60	4990	4.895	4.978	0.544	0.553
3.4	0.559	0.419	71	6272	5.627	5.724	0.625	0.636
3.6	**0.852**	**0.418**	**86**	**8110**	**5.453**	**6.154**	**0.606**	**0.684**
3.8	1.282	0.496	93	8174	5.450	5.808	0.606	0.645
4.0	1.304	0.414	91	9184	5.893	6.206	0.655	0.690
4.5	2.598	0.495	96	11261	5.684	6.536	0.632	0.726
5.0	3.913	0.523	132	15422	5.924	6.906	0.658	0.767
6.0	9.933	0.553	166	22838	5.926	6.897	0.658	0.766
8.0	30.181	0.581	152	32206	6.142	7.680	0.682	0.853
10.0	109.030	0.591	226	56883	5.742	7.660	0.638	0.851

Table 4.6 Parallel metrics of both approaches of Class I on 9 processors

Table 4.6 shows the speedup and efficiency. It also illustrates the ratio of recomputation time to the elapsed time. When the tightness is greater than 3.6 and the ratio of recomputation time to elapsed time is above 41%, then the overhead of state-recomputation model is larger and the run time of state-copying model is faster than that of the state-recomputation model. When the tightness is less than 3.6, the message size of recomputation is quite smaller and recomputation ratio is lower, thus the recomputation performance is better than or equal to the copying performance.

4.4.4 Comparisons on Class I on Multiple Processors

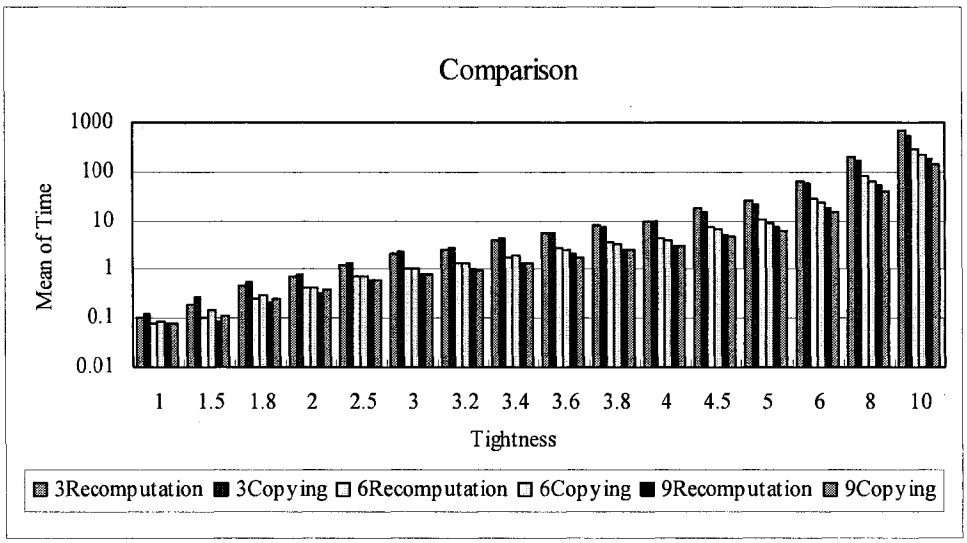


Figure 4.4 Mean of time of Class I on 3, 6, 9 processors

In Figure 4.4, the X axis represents tightness of the problems and the Y axis measures the elapsed times of both approaches on 3, 6 and 9 processors using a logarithmic scale. As the number of processors increase, the elapsed time decreases.

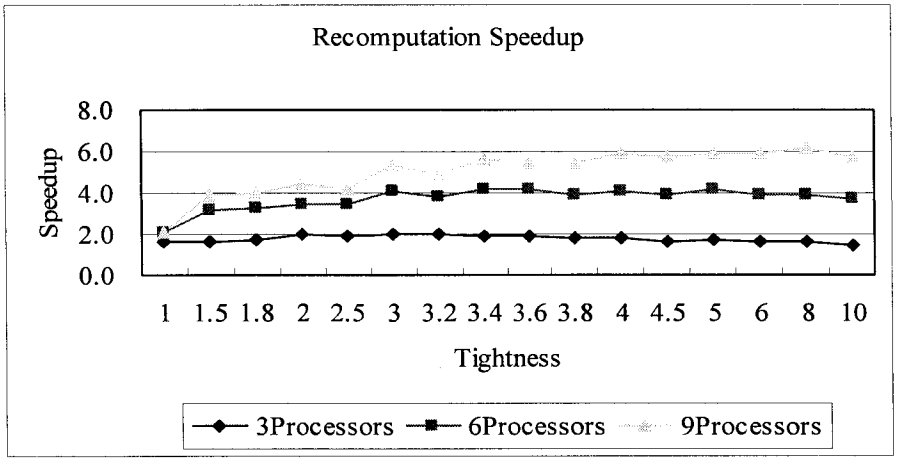


Figure 4.5 Recomputation speedup of Class I on 3, 6, 9 processors

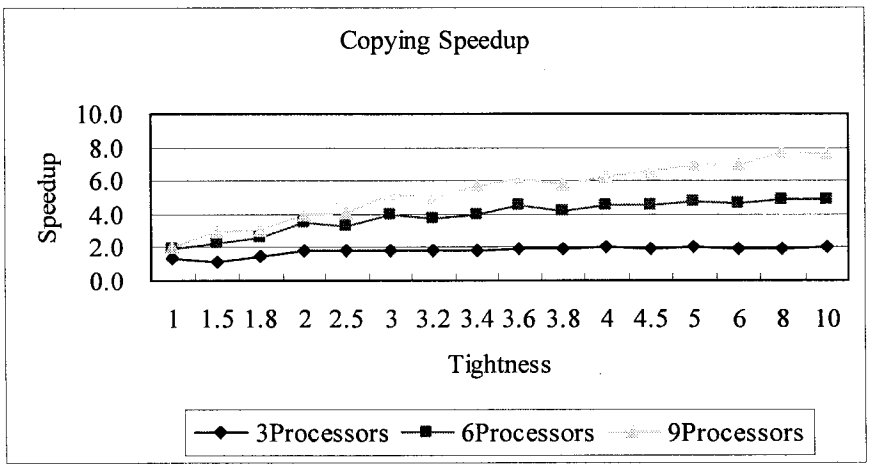


Figure 4.6 Copying speedup of Class I on 3, 6, 9 processors

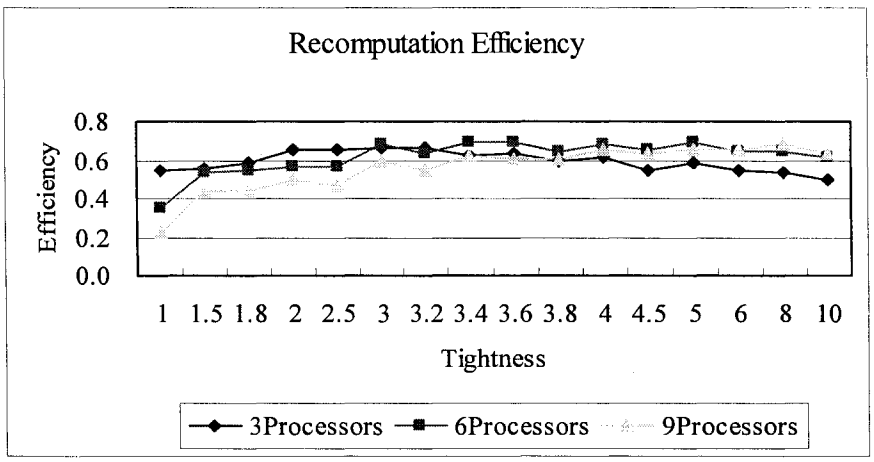


Figure 4.7 Recomputation efficiency of Class I on 3, 6, 9 processors

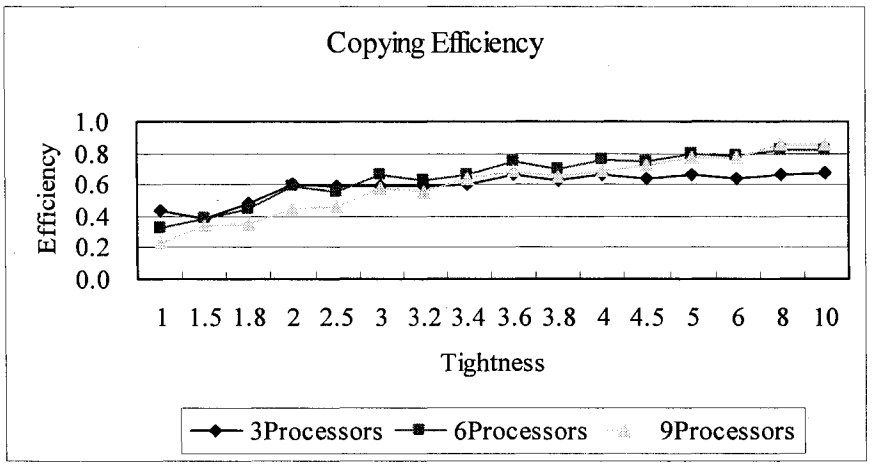


Figure 4.8 Copying efficiency of Class I on 3, 6, 9 processors

Let us take a further look at the parallel metrics of both approaches on 3, 6, and 9 processors. The speedups are noticeable when the number of processors increases. Both approaches show similar characteristics of the speedup. Furthermore, we observe that when the tightness is small, adding processors will result in a loss of efficiency. There is a high overhead to schedule those processors and the extra processors are not being fully utilized. When tightness is larger, the extra processors get used more, resulting in an increase in efficiency. The efficiency depends on the tightness and the number of processors used.

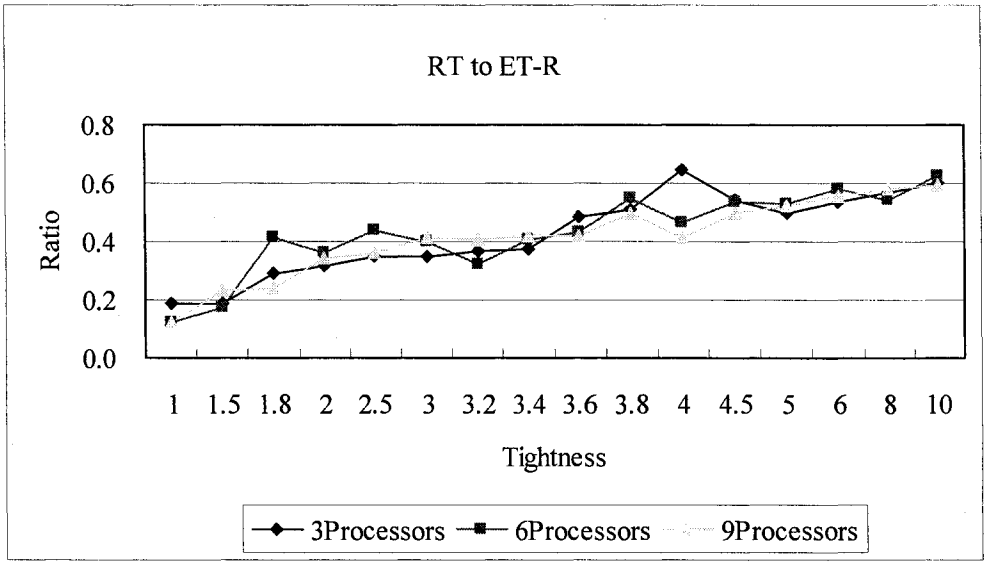


Figure 4.9 Ratio of recomputation time to elapsed time for Class I

Figure 4.9 shows the ratio of time spent on recomputation vs. the problem tightness. In this case, when the tightness is greater than 3.6 and the ratio of recomputation time to elapsed time is above 40%, then the overhead of recomputation-based approach is higher than overhead of the copying-based approach. Thus, the run time of copying-based is shorter than that of the recomputation-based approach. Tables 4.2, 4.4 and 4.6 provide a rough measurement of the communication overhead. When the tightness is less than 3.6 and the ratio is below 40%, the communication cost of copying is greater than the communication cost of recomputation plus the cost of recomputation time. So the recomputation approach will be as, or more efficient than, the copying approach under this situation. Moreover, this kind of performance of both approaches does not change no matter how many the processors we used.

4.4.5 Experimental Results of Class II on 3 Processors

This section discusses experimental results on Class II. This class has 10 variables, domain size is 15, constraint arity is 3 and the constraint density is 0.04. The constraint tightness increases from 1.0 to 6.0.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
1.0	1.983	2.538	-29.695	-0.595	-0.516	Recom
1.1	3.209	3.728	-13.890	-0.598	-0.441	Recom
1.2	4.176	4.789	-11.448	-0.725	-0.500	Recom
1.3	7.382	7.451	-0.953	-0.220	0.083	No diff
1.4	10.032	10.046	-0.118	-0.277	0.248	No diff
1.5	12.872	12.789	0.322	-0.457	0.622	No diff
1.6	17.106	16.367	1.398	-0.371	1.848	No diff
1.7	**25.337**	**23.148**	**5.233**	**1.310**	**3.068**	**Copying**
1.8	27.289	25.931	1.734	-0.287	3.004	Copying
1.9	40.182	35.352	8.319	3.610	6.050	Copying
2.0	49.361	43.424	3.414	2.283	9.590	Copying
2.2	68.658	60.636	3.972	3.779	12.264	Copying
2.4	103.331	85.065	17.554	16.079	20.451	Copying
2.6	153.049	121.488	12.873	26.410	36.712	Copying
2.8	191.291	155.174	18.893	32.101	40.133	Copying
3.0	260.628	202.333	17.031	51.104	65.487	Copying
4.0	1009.846	733.812	23.256	251.098	300.970	Copying
6.0	6220.697	4598.133	26.946	1496.054	1749.074	Copying

Table 4.7 Elapsed time of both approaches of Class II on 3 processors

Table 4.7 contains the elapsed time of both approaches of Class II on 3 processors. The steps represent tightness from 1.0 to 6.0. We use the t-test to compare the two groups. As discussed above, we calculate the T-value and compare it with T_cv = 1.717. In Table 4.7 we also provide the confidence interval.

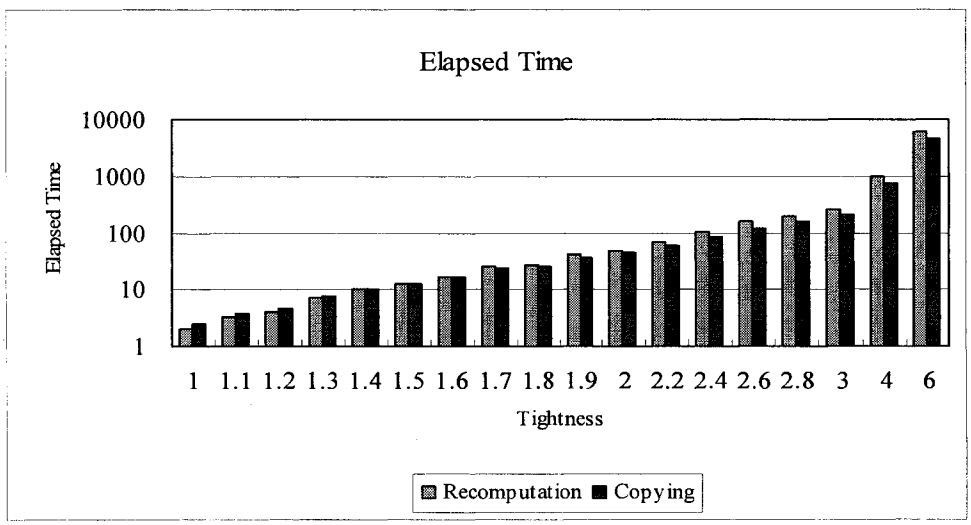


Figure 4.10 Mean of time of Class II for recomputation and copying on 3 processors

We present Figure 4.10 for the equivalent Table 4.7. The X axis represents tightness and the Y axis measures based on a logarithmic scale. In Figure 4.10 we observe that the state-recomputation elapsed time is shorter than or equal to the state-copying when the tightness is less than 1.7. When the tightness is equal to or greater than 1.7, the state-copying elapsed time is faster than the state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
1.0	0.316	0.159	33	4488	1.865	1.457	0.622	0.486
1.1	0.771	0.240	37	5624	1.880	1.619	0.627	0.540
1.2	1.085	0.260	46	6855	1.897	1.654	0.632	0.551
1.3	2.542	0.344	52	7955	1.906	1.889	0.635	0.630
1.4	3.898	0.389	53	9024	2.007	2.004	0.669	0.668
1.5	4.790	0.372	56	10241	1.887	1.899	0.629	0.633
1.6	6.872	0.402	55	11880	1.855	1.939	0.618	0.646
1.7	**11.470**	**0.453**	**57**	**13224**	**1.765**	**1.932**	**0.588**	**0.644**
1.8	11.927	0.437	60	14640	1.952	2.054	0.651	0.685
1.9	18.513	0.461	74	17680	1.773	2.015	0.591	0.672
2.0	23.142	0.469	77	19312	1.608	1.827	0.536	0.609
2.2	32.935	0.480	82	23400	1.732	1.961	0.577	0.654
2.4	51.519	0.499	91	27880	1.573	1.911	0.524	0.637
2.6	78.536	0.513	106	31610	1.635	2.060	0.545	0.687
2.8	99.412	0.520	101	38000	1.564	1.928	0.521	0.643
3.0	135.451	0.520	113	43656	1.600	2.061	0.533	0.687
4.0	568.050	0.563	140	73440	1.406	1.935	0.469	0.645
6.0	3620.850	0.582	242	171494	1.404	1.899	0.468	0.633

Table 4.8 Parallel metrics of both approaches of Class II on 3 processors

Table 4.8 shows the ratio of recomputation time to the elapsed time. In this case when the tightness is greater than 1.7 and the ratio of recomputation time to elapsed time is above 43%, then the overhead of state-recomputation model is higher and the run time of state-copying model is faster than that of the state-recomputation model. When the tightness is less than 1.7, the message size of recomputation is quite smaller and recomputation ratio is lower, so the recomputation performance is better than the copying performance. This table also illustrates the speedup and efficiency.

4.4.6 Experimental Results of Class II on 6 Processors

Table 4.9 contains the elapsed time of both approaches of Class II on 6 processors. When we use the t-test to compare the two groups, we observe that when the |T_value| < T_cv, the tightness is between 1.5 and 1.6, inclusive. We conclude that the two population means times are not significantly different. In Table 4.9 we also provide the confidence interval.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
1.0	0.877	1.099	-21.363	-0.245	-0.201	Recom
1.1	1.351	1.623	-24.913	-0.294	-0.249	Recom
1.2	2.055	2.270	-16.742	-0.242	-0.188	Recom
1.3	3.198	3.292	-2.222	-0.183	-0.005	Recom
1.4	4.221	4.344	-2.805	-0.215	-0.031	Recom
1.5	5.515	5.576	-1.439	-0.149	0.028	No diff
1.6	7.134	6.976	1.529	-0.059	0.375	No diff
1.7	**10.445**	**9.557**	**8.859**	**0.677**	**1.098**	**Copying**
1.8	11.190	10.523	7.918	0.489	0.843	Copying
1.9	16.244	14.891	13.783	1.146	1.559	Copying
2.0	19.167	17.377	14.636	1.533	2.047	Copying
2.2	27.870	25.108	5.565	1.719	3.805	Copying
2.4	43.123	36.122	13.729	5.929	8.072	Copying
2.6	63.253	50.873	22.955	11.247	13.513	Copying
2.8	79.559	64.173	11.569	12.592	18.180	Copying
3.0	102.995	84.367	24.624	17.039	20.217	Copying
4.0	412.367	303.908	26.994	100.018	116.901	Copying
6.0	2532.710	1851.200	43.952	648.935	714.089	Copying

Table 4.9 Elapsed time of both approaches of Class II on 6 processors

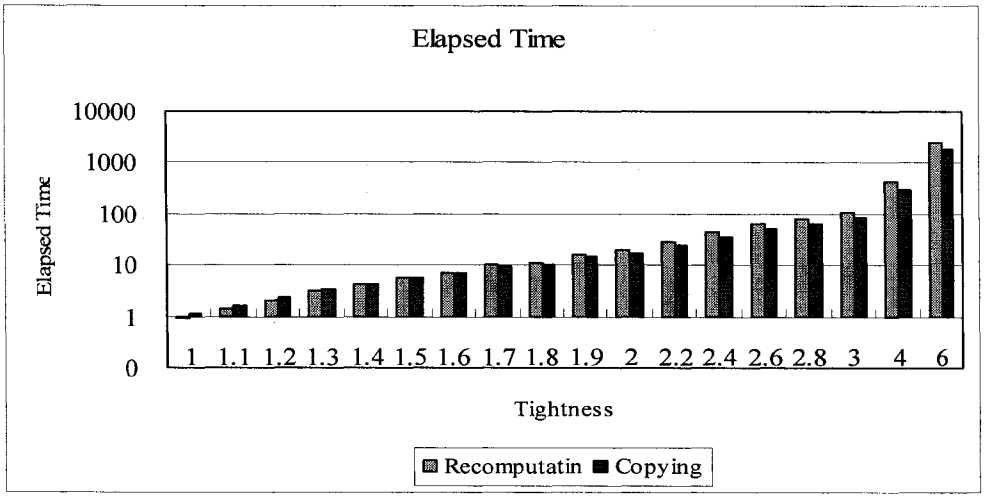


Figure 4.11 Mean of time of Class II for recomputation and copying on 6 processors

We present the X axis represents tightness and the Y axis measures based on a logarithmic scale. Figure 4.11 clearly shows that the state-recomputation elapsed time is faster than or equal to state-copying when the tightness is less than 1.7. When the tightness is equal to or greater than 1.7, the state-copying elapsed time is faster than the state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
1.0	0.311	0.355	33	4488	4.218	3.364	0.703	0.561
1.1	0.485	0.359	37	5624	4.465	3.718	0.744	0.620
1.2	0.665	0.324	55	7413	3.854	3.490	0.642	0.582
1.3	1.186	0.371	67	8624	4.401	4.275	0.734	0.713
1.4	1.602	0.380	57	9638	4.771	4.636	0.795	0.773
1.5	2.156	0.391	61	10853	4.403	4.355	0.734	0.726
1.6	2.948	0.413	73	12760	4.447	4.548	0.741	0.758
1.7	**5.042**	**0.483**	**67**	**14059**	**4.282**	**4.679**	**0.714**	**0.780**
1.8	5.237	0.468	65	15421	4.759	5.061	0.793	0.843
1.9	7.742	0.477	81	17784	4.386	4.785	0.731	0.797
2.0	9.366	0.489	78	19421	4.140	4.567	0.690	0.761
2.2	14.044	0.504	91	23520	4.267	4.736	0.711	0.789
2.4	21.812	0.506	126	31619	3.769	4.500	0.628	0.750
2.6	32.510	0.514	132	35904	3.956	4.919	0.659	0.820
2.8	41.450	0.521	126	39444	3.761	4.663	0.627	0.777
3.0	54.381	0.528	126	44554	4.049	4.943	0.675	0.824
4.0	229.659	0.557	250	92262	3.444	4.673	0.574	0.779
6.0	1474.640	0.582	297	192119	3.448	4.717	0.575	0.786

Table 4.10 Parallel metrics of both approaches of Class II on 6 processors

In this case when the tightness is greater than 1.7 and the ratio of recomputation time to elapsed time is above 46%, the overhead of state-recomputation model is higher and the run time of the state-copying model is faster than that of the state-recomputation model. Table 4.10 also provides when the tightness is less than 1.7, the recomputation performance is better than or equal to the copying performance.

4.4.7 Experimental Results of Class II on 9 Processors

Table 4.11 contains the elapsed time of both approaches of Class II on 9 processors. When we use the t-test to compare the two groups, we observe that when the |T_value| < T_cv, the tightness is between 1.3 and 1.6, inclusive. That means the two population means times are not significantly different. Table 4.11 also provides the confidence interval.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
1.0	0.753	0.761	-0.297	-0.061	0.046	No diff
1.1	1.086	1.132	-1.361	-0.119	0.025	No diff
1.2	1.471	1.637	-9.115	-0.204	-0.128	Recom
1.3	2.422	2.324	1.674	-0.025	0.220	No diff
1.4	3.011	2.946	1.269	-0.042	0.172	No diff
1.5	3.746	3.720	0.930	-0.033	0.085	No diff
1.6	4.890	4.895	-0.082	-0.148	0.137	No diff
1.7	**7.088**	**6.416**	**7.747**	**0.490**	**0.855**	**Copying**
1.8	7.616	7.154	6.239	0.307	0.618	Copying
1.9	10.592	9.487	4.703	0.611	1.597	Copying
2.0	12.567	11.166	9.276	1.084	1.719	Copying
2.2	18.657	15.815	11.354	2.316	3.368	Copying
2.4	28.968	23.584	19.737	4.811	5.957	Copying
2.6	42.053	33.657	12.769	7.015	9.777	Copying
2.8	51.177	41.991	18.202	8.125	10.246	Copying
3.0	66.128	54.033	22.963	10.988	13.201	Copying
4.0	269.453	195.749	23.119	67.006	80.402	Copying
6.0	1602.420	1171.430	41.248	409.046	452.950	Copying

Table 4.11 Elapsed time of both approaches of Class II on 9 processors

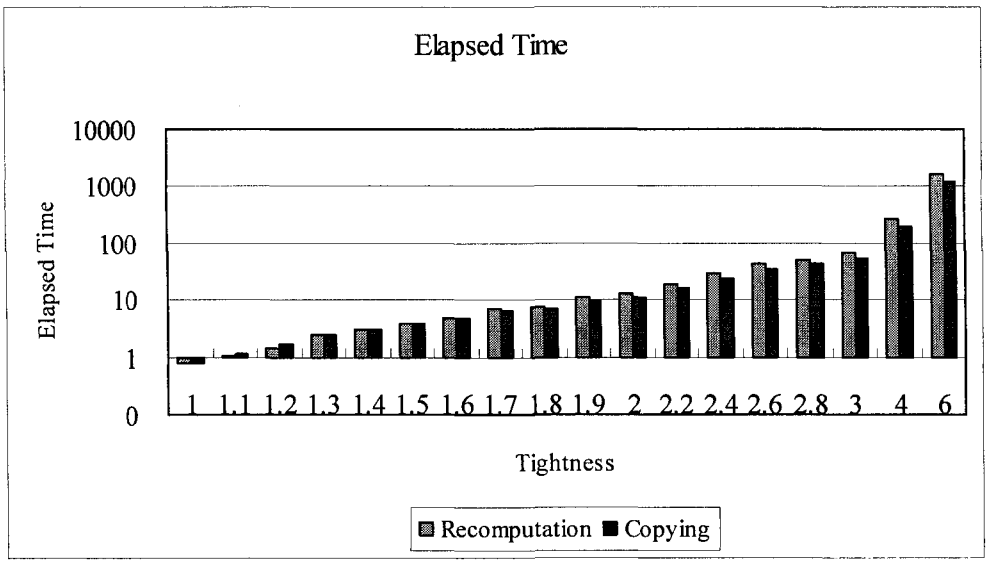


Figure 4.12 Mean of time of Class II for recomputation and copying on 9 processors

Figure 4.12 clearly shows the state-recomputation elapsed time is faster than or equal to the state-copying when the tightness is less than 1.7. When the tightness is equal to or greater than 1.7, the state-copying elapsed time is faster than the state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
1.0	0.264	0.350	39	4542	4.909	4.860	0.545	0.540
1.1	0.368	0.339	43	5685	5.558	5.329	0.618	0.592
1.2	0.483	0.328	63	7511	5.383	4.837	0.598	0.537
1.3	0.973	0.402	80	9539	5.812	6.056	0.646	0.673
1.4	1.133	0.376	71	10253	6.688	6.835	0.743	0.759
1.5	1.621	0.433	72	11342	6.483	6.529	0.720	0.725
1.6	2.103	0.430	72	13552	6.489	6.482	0.721	0.720
1.7	**3.401**	**0.480**	**92**	**14662**	**6.309**	**6.971**	**0.701**	**0.775**
1.8	3.770	0.495	97	16494	6.992	7.444	0.777	0.827
1.9	5.502	0.519	87	18772	6.727	7.510	0.747	0.834
2.0	6.369	0.507	94	19693	6.315	7.107	0.702	0.790
2.2	9.610	0.515	101	24420	6.374	7.519	0.708	0.835
2.4	14.228	0.491	135	32144	5.611	6.892	0.623	0.766
2.6	21.409	0.509	149	38368	5.951	7.435	0.661	0.826
2.8	26.048	0.509	134	41116	5.847	7.126	0.650	0.792
3.0	35.280	0.534	138	45941	6.306	7.718	0.701	0.858
4.0	147.854	0.549	266	97920	5.270	7.255	0.586	0.806
6.0	928.998	0.580	355	206410	5.450	7.455	0.606	0.828

Table 4.12 Parallel metrics of both approaches of Class II on 9 processors

Table 4.12 illustrates the ratio of recomputation time to the elapsed time. When the tightness is greater than 1.7 and the ratio of recomputation time to elapsed time is above 48%, the overhead of the state-recomputation model is higher and the run time of the state-copying model is faster than that of the state-recomputation model. When the tightness is less than 1.7, the recomputation performance is better than or equal to the copying performance.

4.4.8 Comparisons on Class II on Multiple Processors

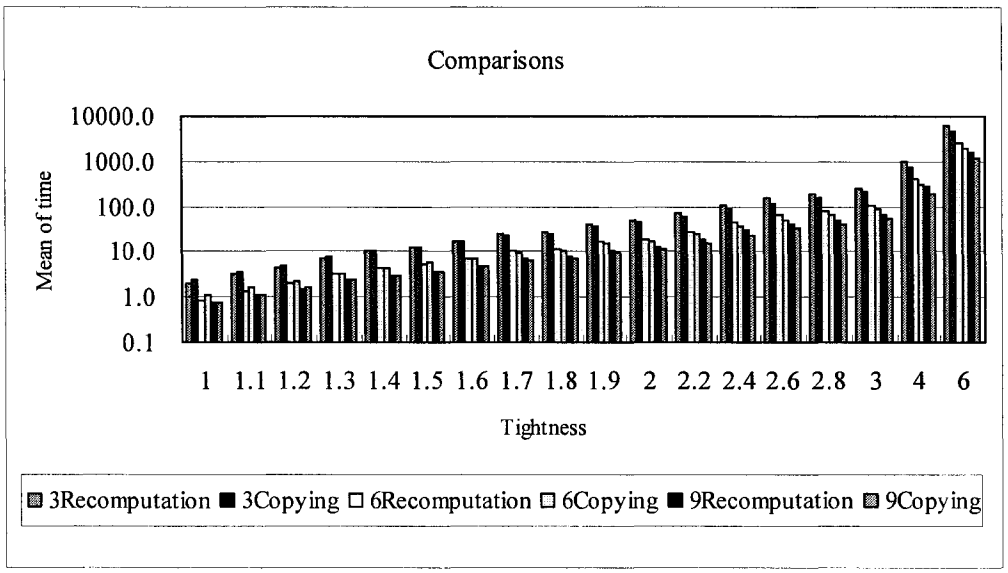


Figure 4.13 Mean of time of Class II on 3, 6, 9 processors

Figure 4.13 shows the X axis represents tightness of problems and the Y axis measures the elapsed times of both approaches on 3, 6, 9 processors in logarithmic scale. As the number of processors increases, the elapsed time decreases on Class II.

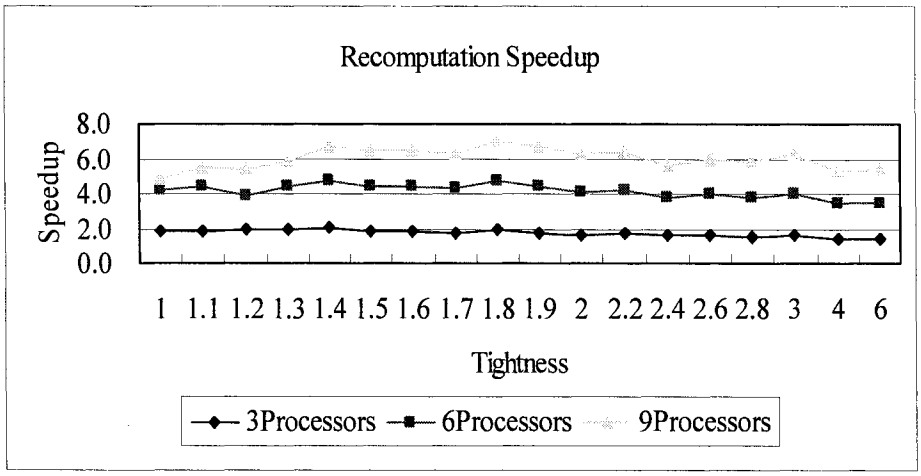


Figure 4.14 Recomputation speedup of Class II on 3, 6, 9 processors

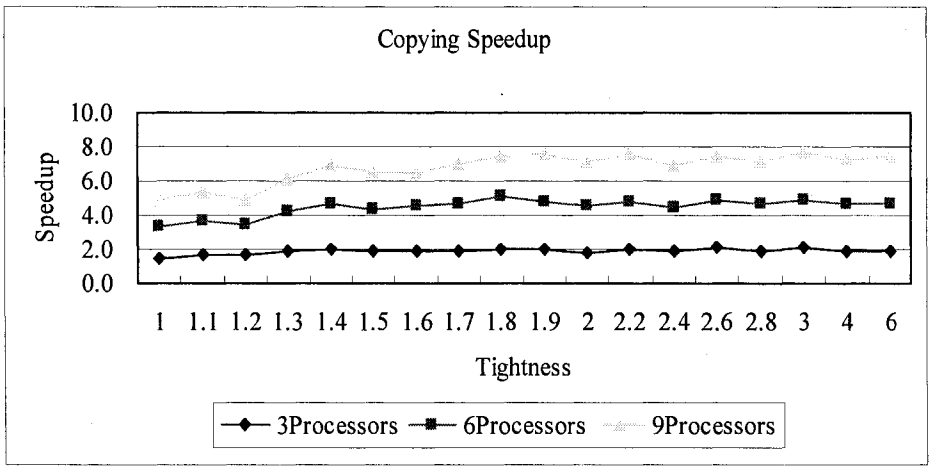


Figure 4.15 Copying speedup of Class II on 3, 6, 9 processors

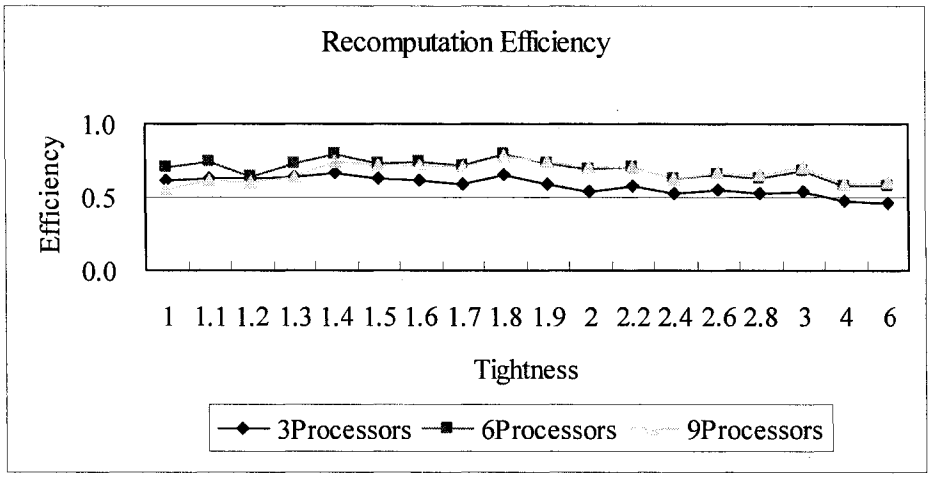


Figure 4.16 Recomputation efficiency of Class II on 3, 6, 9 processors

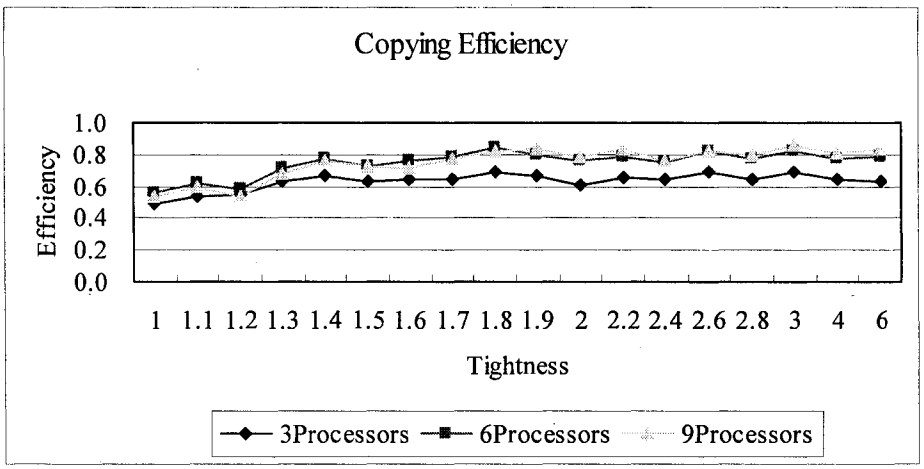


Figure 4.17 Copying efficiency of Class II on 3, 6, 9 processors

Figure from 4.14 to 4.17 are about the parallel metrics of both approaches on 3, 6, and 9 processors. We observe that the more processors involved, the less efficiency achieved on less tight problems. There is higher overhead to schedule those processors and the extra processors are not being fully utilized. But when the tightness is larger, these overheads can be omitted and the efficiency increases with the increasing processors. The efficiency depends on the tightness and the number of processors used. The speedup performs well when the number of processors increases. Both approaches also have similar characteristics with respect to the efficiency and speedup.

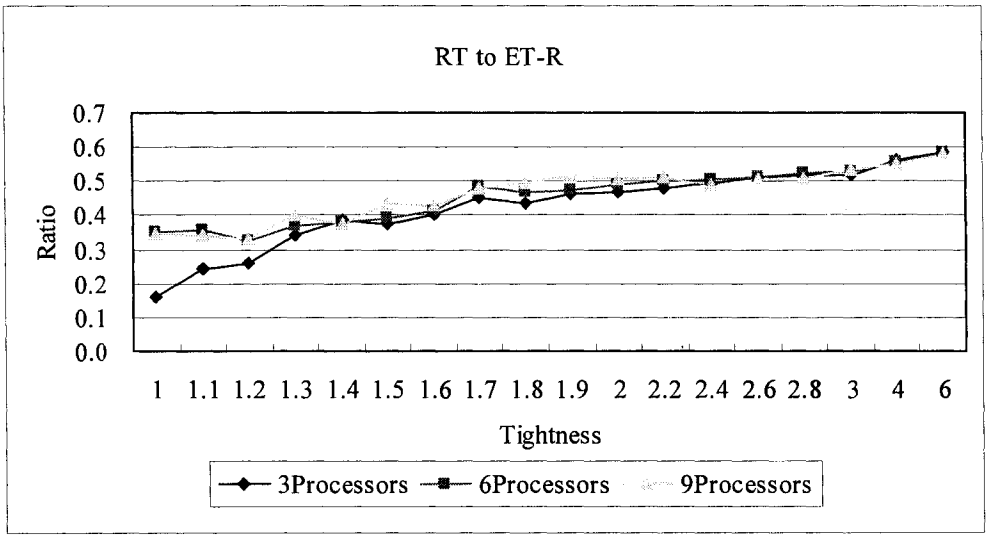


Figure 4.18 Ratio of recomputation time to elapsed time for Class II

Figure 4.18 shows the ratio of recomputation time to elapsed time. In this case, when the tightness is greater than 1.7 and the ratio of recomputation time to elapsed time is above 40%, then the overhead of the recomputation-based approach is higher than the overhead of the copying-based approach; Thus, the run time of copying-based is shorter than that of the recomputation-based approach. When the tightness is less than 1.7, the communication cost of copying is greater than the communication cost of recomputation plus the cost of recomputation time. Therefore, the recomputation approach will be as, or more efficient than, the copying approach in this situation. Furthermore, this kind of performance of both approaches does not alter no matter how many processors we used.

4.4.9 Experimental Results of Class III on 3 Processors

Let us take another class. This section presents experimental results on Class III. This class has 15 variables, domain size is 9, constraint arity is 4 and the constraint density is 0.004. The constraint tightness increases from 0.6 to 2.0.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
0.6	4.050	5.423	-29.026	-1.473	-1.274	Recom
0.7	8.361	10.122	-16.558	-1.984	-1.538	Recom
0.8	10.301	13.999	-22.023	-4.051	-3.346	Recom
0.9	16.334	21.656	-5.316	-7.426	-3.219	Recom
1.0	26.398	30.450	-16.921	-4.555	-3.549	Recom
1.1	39.036	43.314	-9.354	-5.240	-3.318	Recom
1.2	**90.666**	**81.411**	**12.154**	**7.655**	**10.855**	**Copying**
1.3	202.644	164.829	23.022	34.364	41.266	Copying
1.4	448.688	340.523	27.434	99.881	116.448	Copying
1.5	720.406	534.404	27.103	171.584	200.420	Copying
1.6	1254.215	880.368	35.757	351.881	395.813	Copying
1.7	2145.312	1474.840	32.150	626.662	714.290	Copying
1.8	3060.036	2039.440	47.890	975.827	1065.370	Copying
1.9	3714.171	2429.730	43.344	1222.190	1346.700	Copying
2.0	4777.525	3167.900	57.896	1551.220	1668.040	Copying

Table 4.13 Elapsed time of both approaches of Class III on 3 processors

Table 4.13 contains the elapsed time of both approaches of Class III on 3 processors. The steps represent the tightness from 0.6 to 2.0. We calculate the T-value and compare it with T_cv = 1.717. In Table 4.13 we also provide the confidence interval.

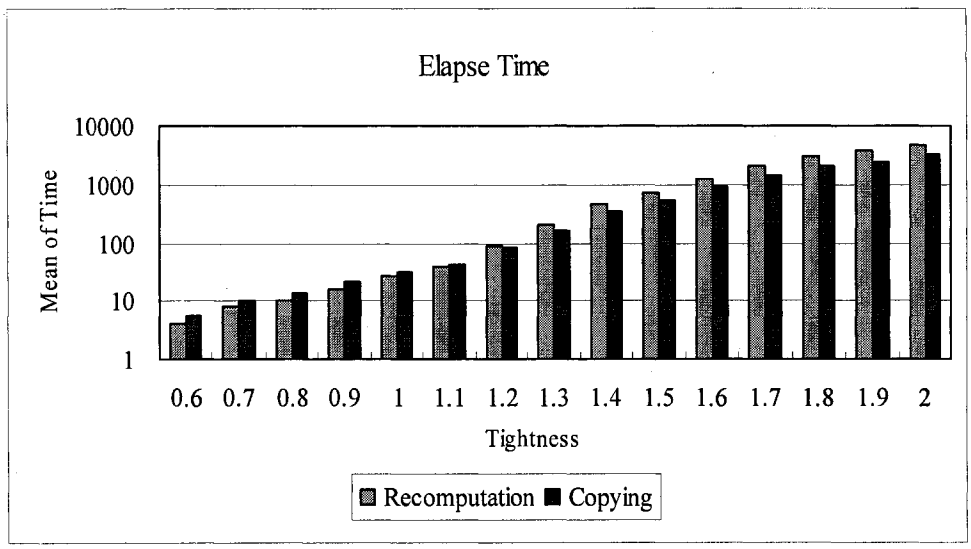


Figure 4.19 Mean of time of Class III for recomputation and copying on 3 processors

We present Figure 4.19 for the equivalent Table 4.13. The X axis represents tightness and the value Y axis measures based on a logarithmic scale. In Figure 4.19 we observe the state-recomputation elapsed time is faster than the state-copying when the tightness is less than 1.2. When the tightness is equal to or greater than 1.2, the state-copying elapsed time is faster than state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
0.6	0.710	0.175	68	9360	1.705	1.273	0.568	0.424
0.7	2.908	0.348	75	13524	1.917	1.583	0.639	0.528
0.8	2.656	0.258	61	16536	1.993	1.467	0.664	0.489
0.9	5.133	0.314	71	21690	1.982	1.495	0.661	0.498
1.0	10.477	0.397	97	25740	1.891	1.639	0.630	0.546
1.1	16.879	0.432	72	31536	2.066	1.862	0.689	0.621
1.2	**45.799**	**0.505**	**148**	**42755**	**1.809**	**2.015**	**0.603**	**0.672**
1.3	121.619	0.600	121	46750	1.567	1.926	0.522	0.642
1.4	279.698	0.623	110	51888	1.532	2.019	0.511	0.673
1.5	453.809	0.630	167	66766	1.524	2.054	0.508	0.685
1.6	824.382	0.657	126	68922	1.393	1.984	0.464	0.661
1.7	1437.730	0.670	376	107654	1.399	2.035	0.466	0.678
1.8	2071.960	0.677	336	107100	1.208	1.812	0.403	0.604
1.9	2549.130	0.686	321	119850	1.081	1.653	0.360	0.551
2.0	3295.780	0.690	303	127670	1.060	1.598	0.353	0.533

Table 4.14 Parallel metrics of both approaches of Class III on 3 processors

Table 4.14 shows the ratio of recomputation time to the elapsed time and speedup and efficiency. In this case when the tightness is greater than 1.2 and the ratio of recomputation time to elapsed time is above 50%, the run time of the state-copying model is faster than that of state-recomputation model. When the tightness is less than 1.2, the recomputation performance is better than the copying performance.

4.4.10 Experimental Results of Class III on 6 Processors

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
0.6	2.564	2.975	-11.756	-0.485	-0.338	Recom
0.7	4.462	5.337	-13.746	-1.009	-0.741	Recom
0.8	5.685	7.022	-11.527	-1.580	-1.093	Recom
0.9	8.743	10.240	-7.585	-1.912	-1.082	Recom
1.0	13.211	14.199	-2.779	-1.734	-0.241	Recom
1.1	18.876	19.612	-2.069	-1.484	0.011	Recom
1.2	**41.078**	**36.328**	**5.510**	**2.939**	**6.561**	**Copying**
1.3	87.189	69.533	23.375	16.069	19.243	Copying
1.4	186.317	142.546	14.870	37.587	49.956	Copying
1.5	301.273	223.812	19.849	69.262	85.660	Copying
1.6	528.491	367.267	22.078	145.882	176.566	Copying
1.7	903.955	611.752	38.840	276.398	308.009	Copying
1.8	1265.530	855.667	55.583	394.371	425.355	Copying
1.9	1496.651	995.930	23.402	455.769	545.674	Copying
2.0	1973.806	1319.250	66.168	633.770	675.336	Copying

Table 4.15 Elapsed time of both approaches of Class III on 6 processors

We use the t-test to compare the two groups and provide the confidence interval.

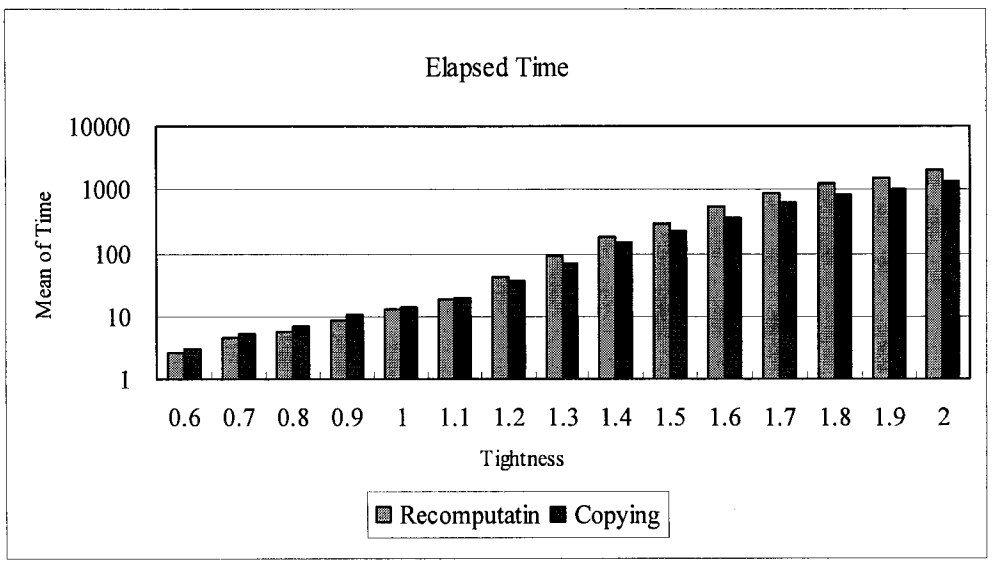


Figure 4.20 Mean of time of Class III for recomputation and copying on 6 processors

Figure 4.20 is equivalent to Table 4.15. We present the X axis is tightness and the value Y axis is based on logarithmic scale. Figure 4.20 clearly shows that the state-recomputation elapsed time is shorter than the state-copying when the tightness is less than 1.2. When the tightness is equal to or greater than 1.2, the state-copying elapsed time is faster than state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
0.6	0.439	0.171	81	11424	2.693	2.321	0.449	0.387
0.7	1.453	0.326	113	17333	3.591	3.002	0.598	0.500
0.8	1.416	0.249	105	20543	3.612	2.924	0.602	0.487
0.9	2.537	0.290	148	28728	3.704	3.162	0.617	0.527
1.0	4.961	0.376	171	34690	3.778	3.515	0.630	0.586
1.1	8.080	0.428	172	39770	4.273	4.113	0.712	0.685
1.2	**22.029**	**0.536**	**280**	**52804**	**3.993**	**4.515**	**0.665**	**0.752**
1.3	53.329	0.612	221	57173	3.641	4.566	0.607	0.761
1.4	119.737	0.643	242	67234	3.690	4.824	0.615	0.804
1.5	191.086	0.634	310	83398	3.644	4.905	0.607	0.818
1.6	345.022	0.653	418	103824	3.305	4.756	0.551	0.793
1.7	604.370	0.669	661	149856	3.320	4.906	0.553	0.818
1.8	853.902	0.675	669	167790	2.920	4.319	0.487	0.720
1.9	1022.610	0.683	540	159750	2.683	4.032	0.447	0.672
2.0	1362.260	0.690	724	201326	2.565	3.838	0.428	0.640

Table 4.16 Parallel metrics of both approaches of Class III on 6 processors

In this case when the tightness is greater than 1.2 and the ratio of recomputation time to elapsed time is above 53%, the overhead of the state-recomputation model is higher and the run time of the state-copying model is faster than that of the state-recomputation model. When the tightness is less than 1.2, the recomputation performance is better than the copying performance.

4.4.11 Experimental Results of Class III on 9 Processors

Table 4.17 contains the elapsed time of both approaches of Class III on 9 processors. When we launch the t-test to compare the two groups, we observe when the |T_value| <

T_cv, the tightness is between 1.0 and 1.1. That means the two population means times are not significantly different. Table 4.17 provides the confidence interval.

Steps	ET-R	ET-C	T_value	CI Lower	CI Upper	Result
0.6	2.245	2.333	-4.049	-0.134	-0.043	Recom
0.7	3.553	4.099	-4.845	-0.782	-0.309	Recom
0.8	4.390	5.156	-4.387	-1.132	-0.399	Recom
0.9	6.862	8.011	-3.849	-1.777	-0.522	Recom
1.0	10.189	10.362	-0.488	-0.922	0.574	No diff
1.1	14.853	14.838	0.029	-1.020	1.049	No diff
1.2	**31.184**	**25.509**	**4.435**	**2.502**	**7.005**	**Copying**
1.3	59.981	47.837	14.182	12.275	16.544	Copying
1.4	127.633	93.019	23.203	31.480	37.748	Copying
1.5	198.857	144.004	19.968	49.082	60.625	Copying
1.6	348.206	242.074	25.775	97.481	114.782	Copying
1.7	601.893	400.308	26.394	185.540	217.631	Copying
1.8	830.345	556.343	30.959	255.408	292.597	Copying
1.9	726.583	488.938	33.683	222.822	252.467	Copying
2.0	1026.203	648.757	9.673	295.466	459.426	Copying

Table 4.17 Elapsed time of both approaches of Class III on 9 processors

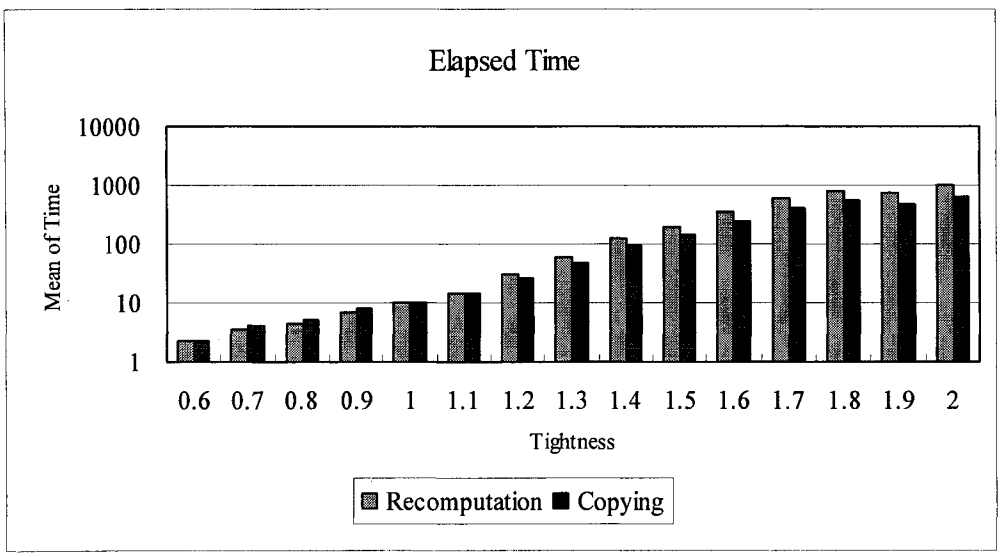


Figure 4.21 Mean of time of Class III for recomputation and copying on 9 processors

Figure 4.21 clearly shows the state-recomputation elapsed time is faster than or equal to the state-copying when the tightness is less than 1.2. When the tightness is equal to or greater than 1.2, the state-copying elapsed time is shorter than the state-recomputation.

Steps	RT	Ratio	MR	MC	SpR	SpC	EffR	EffC
0.6	0.341	0.152	81	11472	3.076	2.959	0.342	0.329
0.7	1.373	0.386	124	18768	4.509	3.909	0.501	0.434
0.8	1.156	0.263	111	22832	4.676	3.982	0.520	0.442
0.9	2.120	0.309	154	30312	4.719	4.042	0.524	0.449
1.0	3.708	0.364	195	38491	4.898	4.816	0.544	0.535
1.1	5.932	0.399	208	45727	5.430	5.436	0.603	0.604
1.2	**15.755**	**0.505**	**339**	**65412**	**5.260**	**6.430**	**0.584**	**0.714**
1.3	36.024	0.601	366	72297	5.293	6.637	0.588	0.737
1.4	79.417	0.622	397	86222	5.387	7.392	0.599	0.821
1.5	128.163	0.644	483	104425	5.521	7.623	0.613	0.847
1.6	227.680	0.654	705	142128	5.017	7.216	0.557	0.802
1.7	402.152	0.668	1002	192461	4.986	7.498	0.554	0.833
1.8	563.234	0.678	910	208916	4.451	6.643	0.495	0.738
1.9	498.922	0.687	1276	252600	5.527	8.214	0.614	0.913
2.0	684.910	0.667	1683	350381	4.934	7.805	0.548	0.867

Table 4.18 Parallel metrics of both approaches of Class III on 9 processors

Table 4.18 illustrates the ratio of recomputation time to the elapsed time. When the tightness is greater than 1.2 and the ratio of recomputation time to elapsed time is above 50%, the run time of the state-copying model is faster than that of the state-recomputation model. When the tightness is less than 1.2, the recomputation performance is better than or equal to the copying performance.

4.4.12 Comparisons on Class III on Multiple Processors

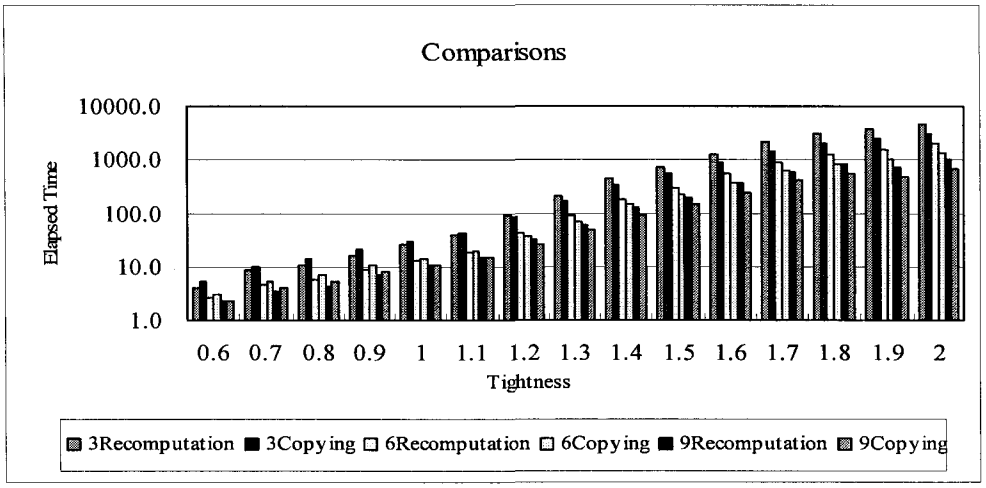


Figure 4.22 Mean of time of Class III on 3, 6, 9 processors

Figure 4.22 shows the X axis represents tightness of problems and the Y axis measures the elapsed times of both approaches on 3, 6 and 9 processors a in logarithmic scale. As the number of processors increases, the elapsed time decreases on Class III.

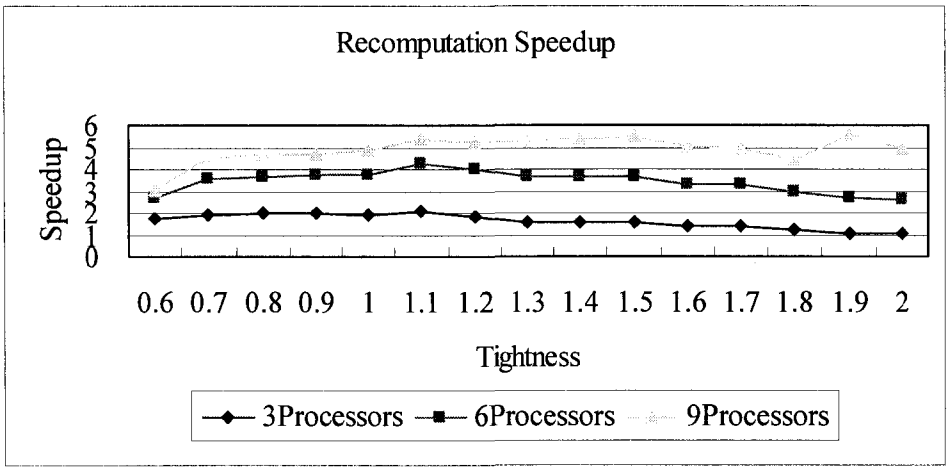


Figure 4.23 Recomputation speedup of Class III on 3, 6, 9 processors

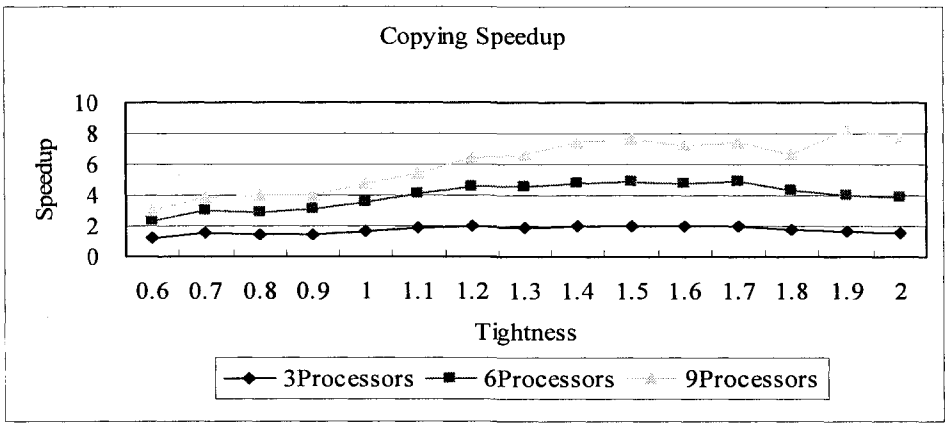


Figure 4.24 Copying speedup of Class III on 3, 6, 9 processors

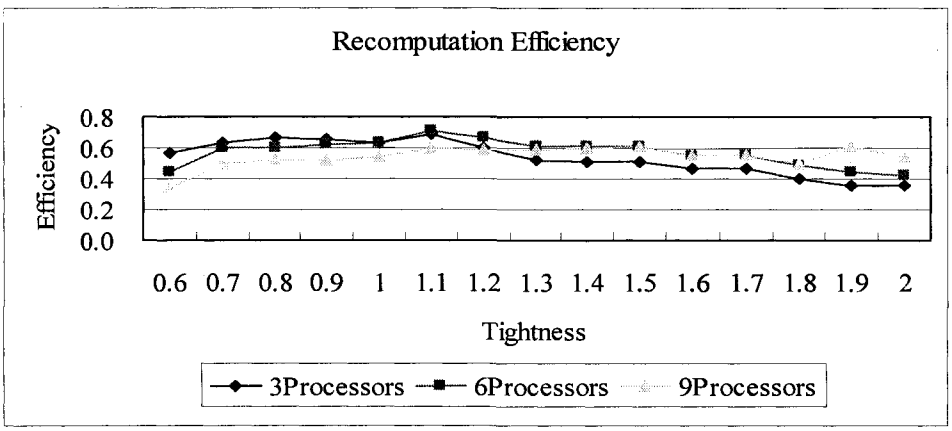


Figure 4.25 Recomputation efficiency of Class III on 3, 6, 9 processors

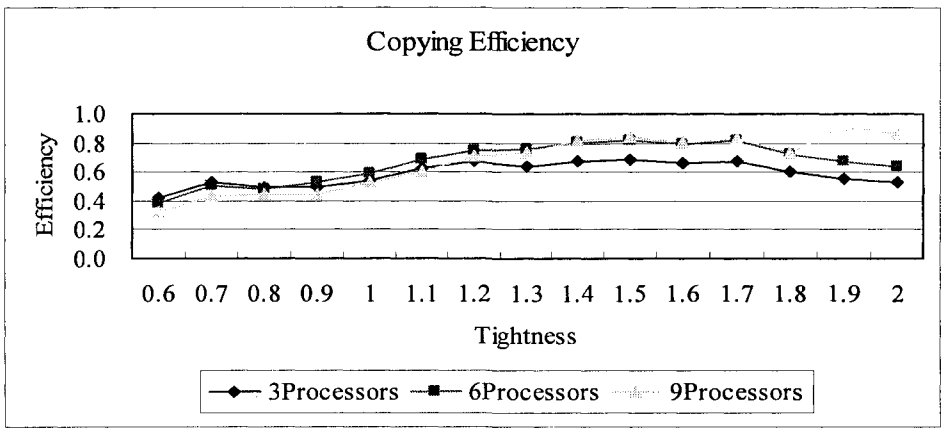


Figure 4.26 Copying efficiency of Class III on 3, 6, 9 processors

Figures 4.23 to 4.26 are about the parallel metrics of both approaches on 3, 6 and 9 processors. We observe that the efficiency does not get good performance with increasing processors on less tight problems. There is higher overhead to schedule those processors. But when the tightness is larger, the efficiency increases with the increasing processors. The efficiency depends on the tightness and the number of processors we used. The speedups achieve good performance when the number of processors increases.

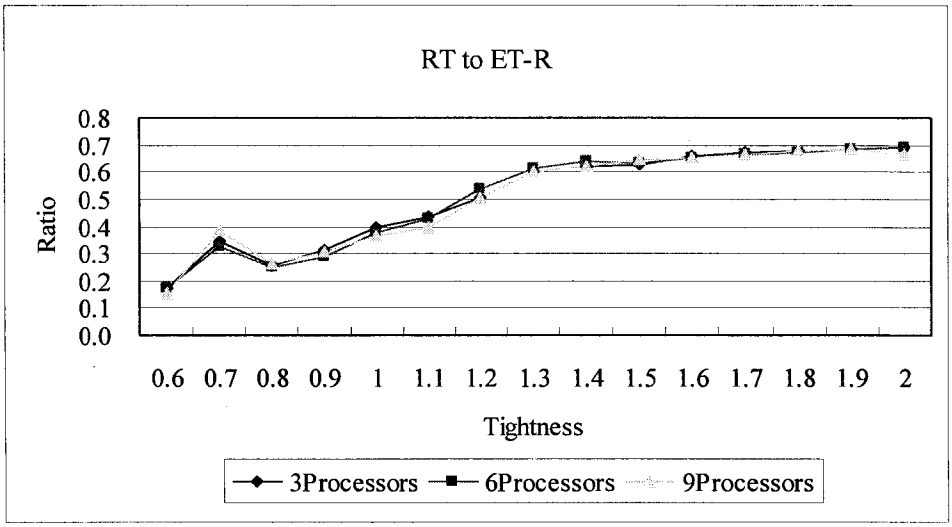


Figure 4.27 Ratio of recomputation time to elapsed time for Class III

Figure 4.27 shows the ratio of recomputation time to elapsed time. When the tightness is greater than 1.2 and the ratio of recomputation time to elapsed time is above 50%, the overhead of the recomputation-based approach is higher than the overhead of the copying-based approach, so the run time of the copying-based is shorter than that of the recomputation-based. When the tightness is less than 1.2, the communication cost of copying is greater than the communication cost of recomputation plus the cost of recomputation time. Thus, the recomputation approach will be as, or more efficient than, the copying approach under this circumstance. Moreover, both approaches performances do not alter no matter how many processors we used.

4.5 Conclusions

This chapter discusses the experimental framework and results. As analyzed above, the copying overhead is mainly due to communication whereas the recomputation overhead is due to communication and recomputation. Based on the experimental data, the analysis and comparison showed that the state-recomputation communication approach will get a better performance when the tightness is lower. By increasing the problem size significantly, the state-copying communication approach will run faster than the other approach. Moreover, this kind of performance of both models does not change no matter how many processors we used.

Chapter 5 Conclusions and Future Work

Parallel computing has been a good candidate for constraint satisfaction problems due to the capability of solving correspondingly larger problems. It can speed up tree searching in order to obtain solutions faster. The parallel search method is a subfield of artificial intelligence in which the constraint satisfaction problem is centralized, while the search processes are distributed among the different processors.

In this thesis we first introduced the background of constraint satisfaction problems, approaches to constraint satisfaction problems, and basic knowledge and techniques of parallel computing. We presented a forward checking algorithm to solve non-binary CSPs by distributing different branches of the search tree to different processors via message passing interface (MPI). We then executed it on the Shared Hierarchical Academic Research Computing Network (SHARCNET). Unfortunately, there existed a communication problem, namely how to efficiently communicate the state of the search among the different processors. In order to solve this communication problem, we proposed two communication models, state-recomputation and state-copying via message passing interface.

We constructed random constraint satisfaction problems based on the standard four-parameter binary model in order to evaluate and compare the behaviour of the two communication models on multiple processors. Finally, we presented empirical results, and statistically analyzed them using the t-test. These experimental results demonstrated that under the same situation, the state-recomputation model with tighter

constraints obtains a better performance than the state-copying model, but when constraints are looser, the state-copying model is a better choice. The characteristics of both approaches were similar no matter how many processors we used. Furthermore, a number of possible factors, including the number of variables, arity of the constraints, constraint density, constraint tightness, and domain size have an impact on which approach should be used.

5.1 Future Work

The experimental study indicated that the main parallel execution overhead of the state-recomputation model is due to recomputation and communication, whereas the overhead of the state-copying model is mainly due to communication. The state-recomputation approach used an oracle to keep track of the search path to guide the recomputation. This definitely has less communication cost than the state-copying approach. The performance of the state-copying is better than the performance of the state-recomputation when the constraint tightness is looser. This is because of the relatively larger percentage of time spent on recomputation. A possible optimization, where a worker is distributed work which requires the least amount of recomputation to reach its current position in the search tree, would be ideal.

In both approaches, when a worker encounters a new choice-point during the execution, it will follow one branch and push other branches into the job queue and then keep going until it branches again or finds a solution. There are two possibilities to request jobs from the job queue. One possibility is that when the worker receives a job request message from the manager, it sends the job back because it has its local queue to work with. The other is that when it finishes its job, it will need the new job from the local job queue. An optimization is as follows: When the worker receives the job request from the manager, it will choose the choice-point that is high in the search tree back to the manager. It possibly contains a large subtree. In this case, it will decrease the network

communication cost. When the worker receives the job request from itself, it will select the adjacent choice-point from the job queue and it can make use of the previous search track. Thus, it will decrease the recomputation time. Under this circumstance, this optimization should be more efficient and intelligent.

Bibliography

[ABJ82] S. G. Akl, D. T. Bernard, and R. J. Jordan, Design and implementation of a parallel tree search algorithm, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4: 192-203, 1982
[Keyword: parallel computing, search algorithm, analysis, implementation, performance, parallel systems, scalability, depth-first search, efficient, overhead]

[AK91] K. A. M. Ali and R. Karlsson, Full prolog and Scheduling Or-parallel in Muse, International Journal of Parallel Programming, 19(6):445-475, 1991
[Keyword: full prolog, scheduling, Or-parallelism, speed up, composition-tree model, re-computing, high-level approach, implicit, independent, logic program]

[Bart98] R. Bartak, Guide to constraint programming, http://kti.ms.mff.cuni.cz/~bartak/constraints/consistent.html, 1998
[Keyword: consistency techniques, node consistency, arc-consistency, path-consistency, k-consistency, binary CSPs, deterministic, non-deterministic, constraint graph, algorithm]

[Bart99] R. Bartak, Constraint Programming: In Pursuit of Holy Grail, In proceedings of Week of Doctoral Student, Prague, 1999
[Keyword: constraint satisfaction, search, inference, combinatorial optimization, planning, scheduling, constraint programming, constraint solving, systematic search, consistency techniques]

[BB98] F. Bacchus, P. Van Beek, On the conversion between Non-binary and binary constraint satisfaction problems, National Conference on Artificial Intelligence, Madison, Wisconsin, 1998
[Keyword: conversion, constraint satisfaction problem, non-binary, binary, dual graph, hidden variable, translation, representation, FC algorithm, overhead, performance]

[BD97] B. Baudot, Y. Deville, Analysis of Distributed Arc-Consistency Algorithms, University catholique de Louvain, Belgium, April 1997
[Keyword: arc-consistency, distributed, shared memory computers, AC-3, AC-4,AC-6, DisAC-3, DisAC-4, DisAC-6, consistency techniques, search space, CSP]

[Beek94] P. Van Beek, On the inherent level of local consistency in constraint networks, In Proceedings of the Twelfth National Conference on Artificial Intelligence, pages 368-373, 1994
[Keyword: local consistency, constraint networks, constraint satisfaction problem, constraint looseness, inherent level, performance, backtracking search, preprocessing, backtrack-free, size of domain]

[Berl95] P. Berlandier, Improving Domain Filtering using restricted path consistency, Proceedings of the IEEE CAIA-95, Los Angeles CA, 1995
[Keyword: path consistency, domain filtering, local consistency, search, solutions, constraint network, arc consistency, inconsistency, k-consistency]

[Bess94] C. Bessiere, Arc-consistency and arc-consistency again, Artificial Intelligence 65, pages 179-190, 1994
[Keyword: arc-consistency, constraint networks, constraint satisfaction problem, AC-4, AC-3, AC-6, space complexity, time complexity, size, drawbacks, binary constraints]

[Bess99] C. Bessiere, Non-Binary Constraints, Principles and Practice of Constraint Programming, Alexandria, Virginia, USA, 1999

[Keyword: non-binary constraints, binary constraints, NP-complete, binary constraint propagation, non-binary constraint propagation, arc-consistency, tree search, generalized arc-consistency, CSPs, backtracking]

[BFR95] C. Bessiere, E. C. Freuder, and J. R. Regin, Using inference to reduce arc consistency computation, In Proceedings of International Joint Conference on Artificial Intelligence, IJCAI-95, pages 592-598, 1995
[Keyword: CSP, constraint, inferences, consistency, AC-7, AC-Inference, constraint checks, arc-consistency, algorithm, scheme]

[BFR99] C. Bessiere, E. C. Freuder, and J. R. Regin, Using constraint metaknowledge and reduce arc consistency computation, Artificial Intelligence 107, pages 125-148, 1999
[Keyword: constraint, constraint metaknowledge, are-consistency, consistency computation, constraint checks, algorithm, CSP, performance, overhead space, time]

[BG95] Fahiem. Bacchus, A. Grove, On the forward checking algorithm, in Proceedings First International Conference on Constraint Programming, Pages 292-308, 1995
[Keyword: CSP, forward checking, backtracking, Backmarking, improvement, minimal forward checking, hybrid, Backjumping, complexity, merits]

[BR75] James R. Bitner and Edward M. Reingold, "Backtrack Programming techniques", Communications of the Association for Computing Machinery, 18(11):651-656, 1975
[Keyword: CSP, backtracking, backtracking programming, programming languages, search, algorithm, recursion, recursive procedure, variables, high-level languages]

[Buyy99] Rajkumar Buyya, High Performance Cluster Computing: Architectures and Systems, Prentice Hall PTR, 1999

[Keywords: high performance, clusters, architectures, parallel programming, supercomputing]

[Cloc92] W.F. Clocksin, The DelPhi Multiprocessor Inference Machine, In proceedings of the 4th U. K. Conference On logic Prog, pages 189-198, 1992
[Keyword: multiprocessor, inference machine, Delphi, logic programming, language, single-processors machine, parallel, performance, efficiency, speedup]

[CS94] P. R. Cooper and M. J. Swain, Arc Consistency: Parallelism and Domain Dependence, Artificial Intelligence, Volume 65: 179-190, 1994
[Keyword: arc-consistency, constraints, parallelism, domain, CSP, parallel constraint satisfaction problem, subspace, search, backtracking, algorithm]

[CSG98] D. E. Culler, J. P. Singh, and A. Gupta, Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann, 1998
[Keyword: parallel programming, programming, performance, shared-memory multiprocessors, design, scalable multiprocessors, cache coherence, tradeoffs, interconnection network, latency tolerance, benchmark]

[DB97] R. Debruyne, C. Bessiere, Some practicable filtering techniques for the constraint satisfaction problem, Proceedings of the 15th IJCAI, pages 412-417, 1997
[Keyword: constraint satisfaction problem, arc consistency, k-consistency, filtering techniques, search, pruning efficiency, qualitative, time efficiency, inconsistency, complexity]

[Dech92] R. Dechter, From local to global consistency, Artificial Intelligence 55, 87-107, 1997
[Keyword: CSP, reasoning, global consistency, local consistency, constraint propagation, instantiation and search tree, arity, constraint network, hidden variables, backtracking]

[Dech98] Rina Dechter, Backtracking algorithms for constraint satisfaction problems, Department of Computer and Information Science, University of California, Irvine, Irvine California, USA,1998
[Keyword: backtracking, look-back, look-ahead, dynamic, variable ordering, constraint satisfaction problems, search, search space, Backjumping, constraint recording, Backmarking, forward checking]

[DF98] R. Dechter, D. Frost, Backtracking algorithms for constraint satisfaction problems: a survey, in Constraints, International Journal, 1998
[Keyword: constraint satisfaction problems (CSPs), Backmarking, search, variable ordering, Backjumping, constraint recording, forward checking backtracking, look-back, look-ahead,]

[DM89] R. Dechter, I. Meiri, Experimental Evaluation of Processing Techniques in Constraint Satisfaction Problems, in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, pages 290-296, 1989
[Keyword: constraint satisfaction problem (CSP), preprocessing, techniques, arc-consistency, path-consistency, forward checking, look-ahead, constraint propagation, backtracking, search tree]

[DMP91] R. Dechter, R. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49, 61-95, 1991
[Keyword: CSP, non-binary constraint, arc consistency, node consistency, constraint propagation, search space, temporal, problem reduction, consistency, backtracking]

[DP88] R. Dechter, J. Pearl, Network-based heuristics for constraint satisfaction problems, Artificial Intelligence 34, 1-38, 1988
[Keyword: CSP, heuristics, hill-climbing, constraint network, search space, propagation, connectionist approach, stochastic search method, consistency]

[DP89] R. Dechter, J. Pearl, Tree clustering for constraint networks, Artificial Intelligence, 38: 353-366, 1989

[Keyword: decomposition, non-binary CSP, binary CSP, tree clustering, triangulation algorithm, graph, width, optimization, transform, primal]

[DR81] A. L. Davis, A. Rosenfeld, Cooperating processes for low-level vision: A survey, Artificial Intelligence, 17, 245-263, 1981
[Keyword: cooperating process, low-level, constraint, constraint satisfaction problem, NP-complete, high-level, performance, search space, complexity]

[FE96] E. Freuder, C. D. Elfe, Neighborhood inverse consistency preprocessing, Proceedings of the AAAI National Conference, pages 202-208, 1996
[Keyword: neighborhood inverse, consistency, arc-consistency, path-consistency, preprocessing, CSP, performance, complexity, search space, algorithm]

[Flyn72] M. J. Flynn, Some computer organizations and their effectiveness, IEEE Transactions on Computers, C-21(9):948–960, 1972
[Keyword: Single instruction, single data (*SISD*), Single instruction, multiple data (*SIMD*), Multiple instruction, single data (*MISD*), Multiple instruction, multiple data (*MIMD*), effectiveness, parallel computing, multiprocessor machine, performance, Flynn's taxonomy, classify]

[Flyn95] M. J. Flynn, Computer Architecture: Pipelined and Parallel Processor Design, Jones and Bartlett, 1995
[Keyword: pipeline, parallel processor, computer architecture, organization, processor design, methodology, evaluation tools, simple queuing theory, probability theory]

[Freu78] E. C. Freuder, Synthesizing constraint expressions, Communications ACM, Vol 21, No. 11, 958-966, November 1978
[Keyword: synthesis, arc-consistency, path consistency, local consistency, CSP, constraint network, binary constraint, non-binary constraint, search space, constraint programming]

[Freu90] E. Freuder, Complexity of K-tree-structured constraint satisfaction problems, In Proceedings of the Eighth National Conference on Artificial Intelligence, page 4-9, 1990
[Keyword: complexity, space, time, structure, search tree, constraint satisfaction problem, constraint network, algorithm, performance, consistency]

[FW89] Eugene C. Freuder, Richard J. Wallace, "Partial Constraint Satisfaction", Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA,1989
[Keyword: Constraint Satisfaction Problem, Variables, Partial constraint satisfaction, backtracking, backjumping, Backmarking, forward checking, Branch and bound, arc-consistency, ordering]

[Gasc74] J. Gaschnig, A constraint satisfaction method for inference making, in Proceedings of the 12th Annual Allerton Conference on Circuit Systems Theory, pages 866-874, 1974
[Keyword: constraint inference, backtracking, arity, problem reduction, binary constraint, arc consistency, node consistency, propagation, search space, CSP]

[Gasc77] J. Gaschnig, A general backtrack algorithm that eliminates most redundant tests, In IJCAI77, Cambridge, MA, Pages 273-312,1977
[Keyword: backtracking, algorithm, redundant, search space, complexity, search space, performance, reduce, test, backtrack free]

[Gasc79] J. Gaschnig, Performance measurement and analysis of certain search algorithms, CMU-CS-79-124 Technical Report, Carnegie-Mellon University, Pittsburg, 1979
[Keyword: performance, analysis, search space, complexity, backtracking, algorithm, measurement, techniques]

[GB65] S. W. Golomb and L. D. Baumert, Backtrack programming, Journal of the ACM 12(4): 516-524, 1965

[Keyword: variables, backtracking programming, programming languages, search, algorithm, recursive procedure, high-level languages, CSP, backtracking, recursion]

[GGK+03] A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Addison Wesley, 2003
[Keyword: parallel computing, parallel programming, MPI, POSIX, OpenMP, Threads, Algorithms]

[GH90] M. L. Ginsberg, W. D. Harvey, Iterative Broadening, AAAI Proceedings, 1990
[Keyword: conventional blind search, distributed, search tree, iterative broadening, heuristic, depth first search, complexity, analysis, artificial breadth cutoff, random]

[Gins93] M. L. Ginsberg, Dynamic Backtracking, Journal of Artificial Intelligence Research 1, pages 25-46, 1993
[Keyword: constraint network, problem reduction, backtracking, dynamic, CSP, binary constraint, backtrack-free, constraint propagation, local consistency, search space]

[GJ79] M. R. Garey and D. S. Johnson, Computers and Intractability: A guide to the theory of NP-Completeness, Freeman, San Francisco, 1979
[Keyword: NP-complete, mathematical, computer, intractability, algorithm, computer science, effective, essence, prove]

[GJC94] M. Gyssens, P. G. Jeavons, and D. A. Cohen, Decomposing constraint satisfaction problems using database techniques, Artificial Intelligence, 66: 57-89, 1994
[Keyword: constraint satisfaction problem, join-dependency, relational database, structure, hypergraph, decomposition, improvement, efficiency, algorithm, subproblem]

[GL96] W. D. Gropp and E. Lusk, User's Guide for MPICH, a Portable Implementation of MPI, Mathematics and Computer Science Division, Argonne National Laboratory, ANL-96/6. 1996

[Keyword: Message passing interface, standard, specification, mpich, parallel, distributed, implementation, symmetric multiprocessors, MPI library, MPI programs]

[GLD+96] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing, 22(6):789–828, September 1996
[Keyword: MPI, specification, MPICH, parallel, portable, library, implementation, high performance, parallel computing, features]

[GLS99] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable parallel programming with the message passing interface, MIT, 1999
[Keywords: message passing, MPI, parallel program, sending message, receiving message, collective computation, communication, portable, high performance, parallel computing]

[GS91] J. Gu, R. Sosic, A parallel architecture for constraint satisfaction, Pages 237-239, 1991
[Keyword: Discrete relaxation algorithm, AI architecture, artificial intelligence, backtracking, backtrack search, constraint satisfaction problem, arc-consistency, algorithm, parallel arc consistency, performance]

[GSN+98] W. Gropp, M. Snir, W. Nitzberg, and E. Lusk, MPI: The Complete Reference, MIT Press, 1998
[Keyword: MPI, message passing, parallel computing, parallel, portable, library, implementation, specification, high performance, features]

[HDR78] R. Haralick, L. S. Dais and A. Rosenfeld, Reduction Operations for Constraint Satisfaction, Information Science 14: 199-219, 1978
[Keyword: backtrack-free, CSP, problem reduction, search, backward checking, constraint propagation, binary constraint, local consistency, constraint network, operations]

[HDT92] P. Van Hentenryck, Y. Deville, and C. M. Teng, A generic arc-consistency algorithm and its specializations, Artificial Intelligence 57, pages 291-321, 1992
[Keyword: instantiation and monotonic, constraint propagation, arc consistency, algorithm, specializations, CSP, local consistency, AC-3, reduction, checking]

[HE80] R. Haralick, G. Elliot, Increasing tree search efficiency for constraint satisfaction problem, Artificial Intelligence 14(3): 263-313, 1980
[Keyword: search tree, performance, complexity, CSP, efficiency, space, time, arity, constraint network, graph]

[Hent89] P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT, Press, 1989
[Keyword: logic programming, constraint satisfaction, prolog, meta-variables, delay declarations, CSP, knowledge, operations, constraint solving, constraint entailment]

[HG95] W. D. Harvey and M. L. Ginsberg, Limited Discrepancy Search, Proceedings of IJCAI95, pages 607-613, 1995
[Keyword: limited discrepancy search, algorithms, tree search methods, heuristics, iterative sampling, backtracking, wrong turns, variations, ordering, local optimization]

[HHM+97] Z. Habbas, F. Herrmann, P. P. Merel, and D. Singer, Load balancing strategies for parallel forward search algorithm with conflict based backjumping, Laboratoire de recherché en Informatique de Metz, Ile du Saulcy, 1997
[Keyword: CSP, NP-complete, backtracking, search algorithm, performance, forward checking, tree-search splitting, parallel, load balancing, binary CSP]

[HKS00] Z. Habbas, Michael Krajecki, Daniel Singer, Domain Decomposition for Parallel Resolution of Constraint Satisfaction Problems with

OpenMP, In Proceedings 2nd, European Workshop on OpenMP, EWOMP 2000, Edinburgh UK, PP. 1-8, 2000
[Keyword: CSP, parallel processing, OpenMP, Domain decomposition, irregular application, search algorithm, filtering strategies, heuristics, subproblem, NP-complete, shared memory architecture]

[HL88] C. Han and C. Lee, Comments on Mohr and Henderson's path consistency algorithms, Artificial Intelligence 36, pages 125-130, 1988
[Keyword: path consistency, node consistency, arc consistency, CSP, algorithm, efficiency, Mohr and Henderson's path consistency, performance, search space, arity]

[HX98] K. Hwang and Z. Xu, Scalable Parallel Computing, McGraw-Hill, New York, NY, 1998
[Keyword: multiprocessors, concurrent programming, models of computation, communication, scalable, parallel computing, message passing, principles, approaches, architecture, algorithms, programming languages]

[JMJ96] J. Jaffar, M. J. Maher, J, Constraint Logic Programming---A Survey, 19/20:503-581, 1996
[Keyword: constraint logic programming, constraint solving, logic programming, variables, constraints, algorithms, backtracking, application, implementation, CLP languages]

[Kasi90] S. Kasif, On the parallel complexity of discrete relaxation in constraint satisfaction networks, Artificial Intelligence, 45: 275-286, 1990
[Keyword: parallel computing, complexity, discrete relaxation, CSP, constraint network, performance, local consistency, optimization, search space, discrete]

[KB97] G. Kondrak, and P. van Beek, A theoretical evaluation of selected backtracking algorithms, IJCAI, In proceedings international joint conference on artificial intelligence, 1997

[Keyword: backtracking (BT), constraint network, search tree, CSP, inconsistency, instantiation, variables, backjumping, conflict-directed backjumping, hierarchies]

[KGR94] V. Kumar, A. Grama, and V. N. Rao, Scalable load balancing techniques for parallel computers, Journal of Parallel and Distributed Computing, 22(1):60– 79, July 1994
[Keyword: parallel computing, scalable, load balancing, algorithms, partitioning, processors, schemes, architecture, accuracy and viability, analysis]

[KR87] V. Kumar and V. N. Rao, Parallel depth-first search, part II: Analysis. International Journal of Parallel Programming, 16(6):501–519, 1987
[Keyword: parallel, depth-first search, backtracking, dynamic, static, load balancing, work-splitting, schemes, asynchronous round robin, global round robin, random polling]

[Kuma92] Vipin Kumar, "Algorithms for Constraint Satisfaction Problem: A Survey", AI Magazine, 13(1):32-44, 1992
[Keyword: Artificial Intelligence, Constraint Satisfaction Problem, constraint propagation, backtracking, combination, binary CSP, Arc Consistency, Performance, General CSP, Intelligent Backtracking]

[Mack77] A. K. Mackworth, Consistency in networks of relations, Artificial Intelligence 8, pages 99-118, 1977
[Keyword: constraint graph, consistency, local consistency, constraint networks, propagation, binary constraint, arc consistency, node consistency, relation, artificial intelligence]

[Mess94] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Available at http://www.mpi-forum.org May 1994
[Keyword: MPI, message passing, specification, standard, parallel, distributed, implementation, symmetric multiprocessors, MPI library, MPI program]

[Mess97] Message Passing Interface Forum, MPI-2: Extensions to the Message-Passing Interface, Available at http://www.mpi-forum.org July 1997
[Keyword: message passing, specification, parallel, implementation, standard, symmetric multiprocessors, distributed, MPI library, MPI program, MPI]

[MF85] A. K. Mackworth and Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, Artificial Intelligence 25, pages 65-74, 1985
[Keyword: network, artificial intelligence, CSP, complexity, arc-consistency, local consistency, k-consistency, algorithms, search tree, reduction]

[MH86] R. Mohr and T. C. Henderson, Arc and path consistency revised, Artificial Intelligence 28, pages 225-233, 1986
[Keyword: local consistency, arc-consistency, path-consistency, constraint propagation, instantiation, search tree, CSP, reduction, backtracking, consistency]

[Mont74] U. Montanari, "Network of constraint: fundamental properties and applications to picture processing", Information Science, 7:95-132, 1974
[Keyword: constraint, constraint satisfaction problem, constraint network, properties, applications, picture processing, performance, complexity, constraint graph, fundamental]

[MS94] S. Mudambi, J. Schimpf, Parallel CLP on Heterogeneous Networks, European Computer-Industry Research Centre, 1994
[Keyword: parallel, logic programming, CLP, heterogeneous computing, recomputation, stack-copying, stack-recomputation, Or-parallelism, network, model]

[MS98] K. Marriott, P. J. Stuckey, Programming with Constraint: An introduction, MIT Press, 1998

[Keyword: programming, constraint, CSP, constraint logic programming, constraint programming, constraint imperative programming, concurrent constraint programming, consistency, CP, applications]

[Nade88] B. Nadel, Tree search and arc consistency in constraint satisfaction algorithms, in Search in Artificial Intelligence, Springer-Verlag, pages 287-342, 1988

[Keyword: tree search, arc consistency, constraint satisfaction problems, CSP, node consistency, constraint propagation, backtracking, forward checking, algorithms, complexity]

[Naga01] Sivakumar Nagarajan, On dual encodings for constraint satisfaction, PhD Thesis, University of Regina, 2001

[Keyword: constraint graph, problem reduction, search, synthesis, arc consistency, consistency, algorithms, constraint coverings, dual encoding, unary constraint, preprocessing]

[ND95] T. Nguyen, Y. Deville, A distributed arc-consistency algorithm, Science of Computer Programming, 1995

[Keyword: constraint techniques, CSP, arc-consistency, AC-4, DisAC-4, parallel algorithms, distributed memory computers, speedup, complexity, sequential algorithm]

[NG] S. Nagarajan, Scott. D. Goodwin, A randomized parallel approach to synthesis based constraint satisfaction, Dept. of Computer Science, University of Regina, Canada

[Keyword: synthesis, random, parallelism, CSP, CDGT, algorithms, constraint network, parallel computation, space complexity, performance]

[Nide83a] Nidel, B.A, Consistent-labelling problems and their algorithms, Proceedings National Conference on Artificial Intelligence (AAAI), pages 128-132, 1982

[Keyword: consistency, arc-consistency, algorithm, complexity, space, time, consistent-labelling, artificial intelligence, applications, performance]

[Nide83b] Nidel, B. A, Solving the general consistent labelling (or constraint satisfaction) problem: two algorithms and their expected complexities, Proceedings National Conference on Artificial Intelligence (AAAI), pages 292-296, 1983
[Keyword: CSP, complexity, algorithms, consistency, arc-consistency, space, consistency-labelling, artificial intelligence, applications, performance]

[Nilm91] L. M. Ni, A layered classification of parallel computers, In Proceedings of 1991 International Conference for Young Computer Scientists, 28–33, 1991
[Keyword: classification, SIMD, MIMD, SISD, MISD, parallel computing, architecture, speedup, performance, high-level, algorithm]

[OMP97] OpenMP Architecture Review Board, OpenMP C and C++ Application Program Interface, Oct. 1997, http://www.openmp.org/
[Keyword: OpenMP, API, shared memory machine, parallel programming, parallel applications, various platforms, supercomputers, architecture, flexible, interface]

[P97] W. Pang, Constraint-Directed Approach for Analyzing and Solving General Constraint Satisfaction Problems, PhD thesis, University of Regina, Saskatchewan, Canada, 1997
[Keyword: CDBT, backtracking, algorithm, variable set, domain, performance, search space, general CSP, partial solution, subset]

[Pach98] P. Pacheco, Parallel Programming with MPI, Morgan Kaufmann, 1998
[Keyword: parallel programming, MPI, message passing, specification, libraries, point-to-point communication, collective communication standard, implementation, applications]

[Perl92] M. Perlin, Arc consistency for factorable relations, Artificial Intelligence 53, pages 329-342, 1992
[Keyword: arc consistency, constrain graph, search space, complexity, CSP, artificial intelligence, reduction, relations, path consistency, constraint network]

[PG97a] Wanlin Pang, Scott D. Goodwin, Constraint-Directed Backtracking, Australian Joint Conference on Artificial Intelligence, 1997
[Keyword: CDBT, BT, naïve backtracking, variable set, domain, search space, general CSP, performance, algorithm, partial solution]

[PG97b] W. Pang, Scott D. Goodwin, A revised sufficient condition for backtrack-free search, In proceedings of 10th Florida AI Research Symposium, Pages 52-56, Daytona Beach, FL, May 1997
[Keyword: w-consistency, w-graph, CSP, backtracking, general CSP, backtrack-free, decomposition, w-k-consistency, constraint-directed backtracking, binary CSP]

[PSW00] P. Prosser, K. Stergiou, T. Walsh, Singleton Consistencies, Proceedings Principles and Practice of Constraints Programming, pages 353-368, 2000
[Keyword: singleton consistencies, preprocessing, algorithm, random, structured, consistency techniques, arc-consistency, reduce, complexity, local consistencies]

[RDP90] F. Rossi, V. Dahr, C. Petrie, On the equivalence of constraint satisfaction problems, European Conference on Artificial Intelligence, Stockholm, 1990
[Keyword: CSP, solution, mutual reducibility, equivalence, non-binary CSPs, transformation, binary CSPs, variables, dual representation, constraint]

[Regi94] J. C. Regin, A filtering algorithm for constraints of difference in CSPs, Proceedings of the Twelfth National Conference on Artificial Intelligence, pages 362-367, 1994

[Keyword: filtering algorithms, constraints, CSP, general CSP, reduction, search space, complexity, performance, computing, constraint graph]

[RK94a] V. N. Rao, and V. Kumar, Parallel depth first search, Part 1: Implementation, Department of computer science, University of Texas at Austin, 1994

[Keyword: implementation, depth-first search, backtracking, dynamic, static, load balancing, work-splitting, schemes, asynchronous round robin, global round robin, random polling]

[RK94b] V. R. Rao and V. Kumar, Parallel depth first search, Part 2: Analysis, Department of computer science, University of Texas at Austin, 1994

[Keyword: analysis, backtracking, dynamic, static, load balancing, work-splitting, schemes, asynchronous round robin, global round robin, random polling, depth-first search]

[Sand95] Peter Sanders, Better Algorithms for Parallel Backtracking, Workshop on Parallel Algorithm for Irregularly Structured Problems, 1995

[Keyword: artificial intelligence, depth first search, parallelism, communication, load balancing, parallel computing, backtracking, processors, distribute, search tree]

[Seid81] R. Seidel, A new method for solving constraint satisfaction problems, Proceedings 7th International Joint Conference on AI, pages 338-342, 1981

[Keyword: Invasion algorithm, solutions, synthesis algorithm, search tree, partial graphs, general CSP, binary CSP, Essex algorithms, performance, complexity]

[SH87] A. Samal and T. C. Henderson, Parallel consistent labeling algorithms, Int. J. Parallel Program, 16: 341-364, 1987

[Keyword: Parallel algorithm, arc consistency, AC-1, AC-3, AC-4, performance, shared memory, design, parallel, implementation]

[Shap89] E. Shapiro, Or-Parallel Prolog in Flat Concurrent Prolog, In Journal of logic programming, pages 243-267, 1989
[Keyword: multiple processors, parallel computing, stack-copying, stack-recomputation, reduce, communication, overhead, oracle, worker, manager]

[Shar01] SHARCNET forum, SHARCNET: The Shared Hierarchical Academic Research Computing Network (SHARCNET), Available at http://www.sharcnet.ca, June, 2001
[Keyword: HPC, SHARCNET, high tech, distributed system, accelerate, high performance, parallel computing, unique model, hardware, software]

[Sing95] M. Singh, Path consistency revised, Proceedings of the 7th IEEE International Conference on Tolls with Artificial Intelligence, pages 318-325, 1995
[Keyword: path consistency, consistency, arc-consistency, constraint network, CSP, binary CSP, k-consistency, generalized arc consistency, search tree, reduction]

[Smit94] B. M. Smith, The phase transition and the mushy region in constraint satisfaction problems, European conference on artificial intelligence (ECAI-94), pages 100-104, 1994
[Keyword: CSP, randomly-generated, phase transition, tightness, constraint density]

[Sosi94] Rok Sosic, A parallel search algorithm for the N-Queen problem, parallel computing and transputer conference, Wollongong, pages 162-172, Nov, 19940
[Keyword: constraint satisfaction, parallel search, local search, n-queens problem, algorithm, sequential algorithm, data structure, probabilistic parallel search, CSPs, processors]

[SW99] K. Stergiou, T. Walsh, Encodings of Non-binary Constraint Satisfaction Problems, National Conference on Artificial Intelligence, Orlando, Florida, 1999
[Keyword: encoding, non-binary CSP, binary CSP, constraint satisfaction problem, arc consistency, dual, hidden variable, path consistency, constraint graph, mapping]

[TBK95] Edward P. K. Tsang, James E. Borrett, Alvin C. M. Kwan, An attempt to map the performance of a range of algorithm and heuristic combinations, Department of Computer Science, University of Essex, Wivenhoe Park, United Kingdom, 1995
[Keyword: constraint satisfaction problem, AI, algorithm, heuristic, implication, application domain, effective, combinations, range, MAC, FC_CBJ]

[TF90] E. Tsang, N. Foster, Solution synthesis in the constraint satisfaction problem, Technical Report CSM-142, Dept. of Computer Science, University of Essex, 1990
[Keyword: solution synthesis, CSP, general CSP, binary CSP, legal compound labels, Freuder's algorithm, invasion algorithm, Essex algorithm, parallel, searching]

[TK93] Edward Tsang, Alvin Kwan, Mapping Constraint Satisfaction Problems to Algorithms and Heuristics, Department of Computer Science, University of Essex, 1993
[Keyword: CSP, chronological backtracking, branch and bound, forward checking, lookahead, arc-consistency, backjumping, solution synthesis, variables ordering, local search, genetic algorithm, connectionist methods]

[Tsan93] E. Tsang, Foundations of Constraint Satisfaction, Academic Press
[Keyword: constraint satisfaction problem, problem reduction, searching, backtracking, solution synthesis, consistency, forward checking, hill-climbing, heuristic, variable, domain, constraint]

[Ullm76] J. R. Ullman, An algorithm for Subgraph Isomorphism, Journal of the ACM 23:31-42, 1976
[Keyword: Subgraph isomorphism, brute-force, tree-search, algorithm, random, nonrandom, implementation, efficiency, performance, time, space]

[VK86] M. Villain, H. Kautz, Constraint-propagation algorithms for temporal reasoning, In Proceedings of the Fifth National Conference on Artificial Intelligence, pages 377-382, 1986
[Keyword: Constraint propagation, constraint graph, temporal reasoning, CSP, reduction, algorithms, search tree, search space, complexity, constraint network]

[VMB99] G. Verfaillie, D. Martinez, and C. Bessiere, A generic customizable framework for inverse local consistency, Proceedings of the AAAI National Conference, pages 169-174, 1999
[Keyword: local consistency, consistency, CSP, arc consistency, path consistency, k consistency, (i,j)-consistency, inverse local consistency, general framework, AC-7, algorithm]

[YH00] Makoto Yokoo, Katsutoshi Hirayama, "Algorithms for Distributed Constraint Satisfaction: A Review", Autonomous Agents and Multi-Agent Systems, 2000
[Keyword: Constraint Satisfaction, Search, Distributed AI, Distributed CSP, Multi-agent, Asynchronous Backtracking, Asynchronous weak-commitment search, Distributed consistency, Algorithms]

Vita Auctoris

Jigang Yang was born in Shanghai, China. He graduated from Shanghai Normal University No. 3 Fu Zhong High School. He obtained a B.Eng. in Computer Science and Technology in Tsinghua University in 1997. After that he worked at VERITAS Software Corporation. He is currently a candidate for the Master's degree in Computer Science at the University of Windsor and hopes to graduate in Fall 2004.